TELECOMMUNICATIONS
JUNE 2025
ISSUE 419

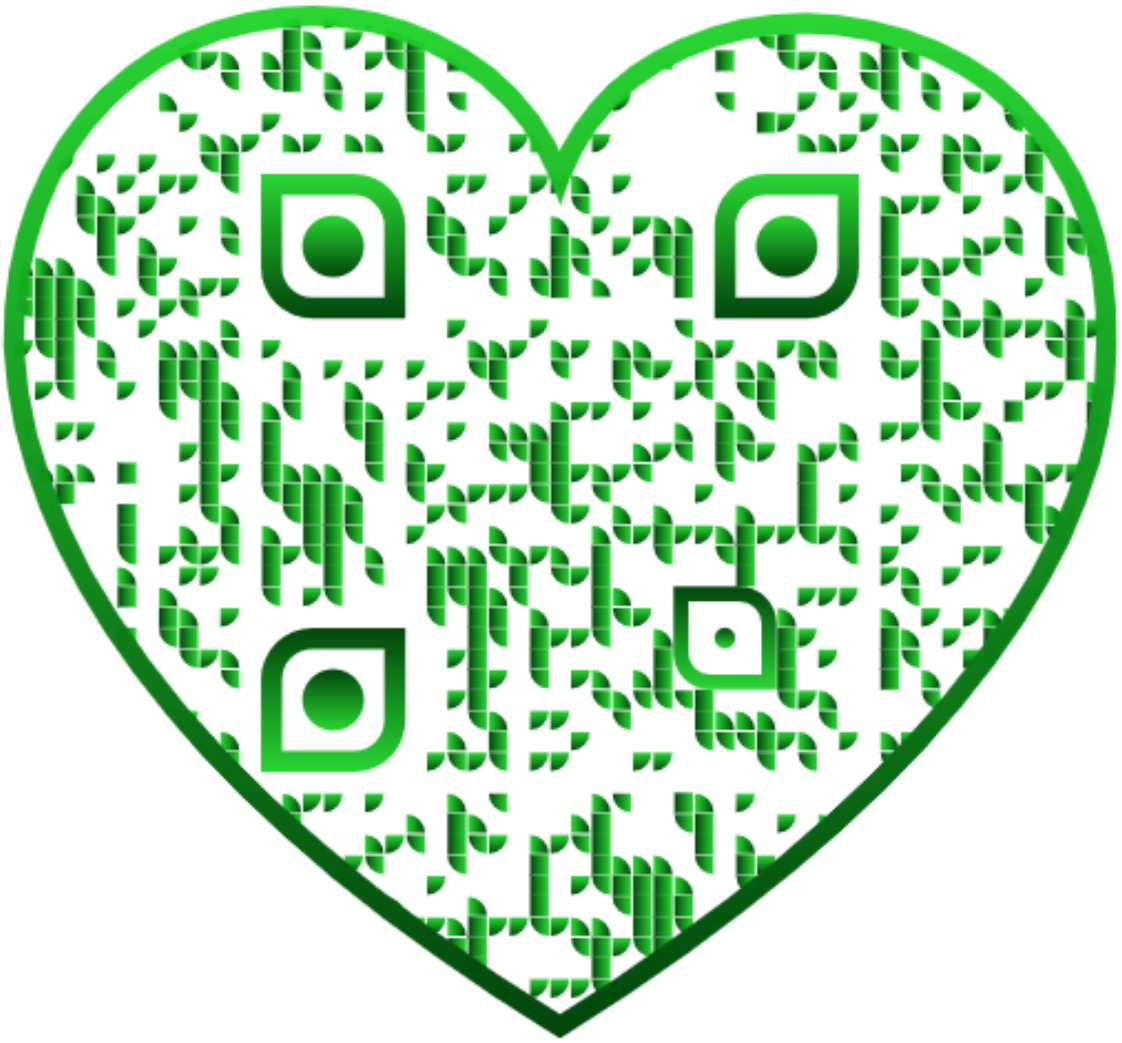
circuitcellar.com

circuit cellar
Inspiring the Evolution of Embedded Design

TELECOMMUNICATIONS

$9.00US $10.00CAN
06
0 74470 75349 0

- AC-DC Power Supplies
- Edge Computing Paradigm |
- AI-Driven Smart Home Face Recognition |
- Zero Trust Architecture | ESP32 Control System for Aquariums
- Improve Portability and Reusability | Releasing Your Flutter App |
- Redux GPS Crystal Calibration
- The Future of Graphical User Interfaces



Imagine what you will

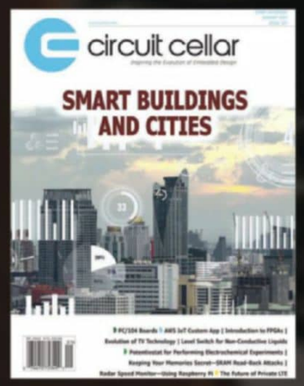
Discover

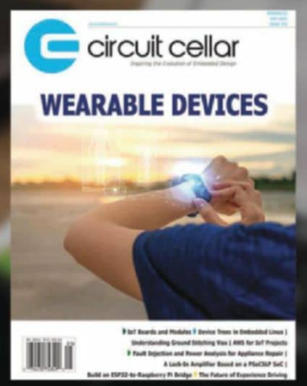
400+ ISSUES

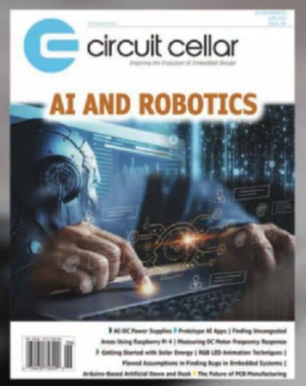
CC VAULT

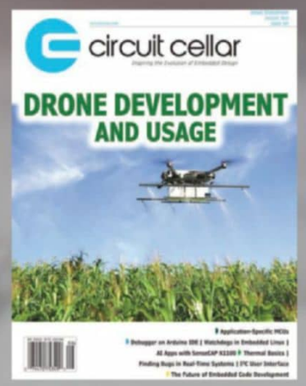
Get the complete *Circuit Cellar* issue archive and article code stored on a stylish, durable and portable USB flash drive. You can easily keep your CC Vault archive up to date by purchasing subsequent issues from our webshop or by downloading issues with a *Circuit Cellar* Digital Subscription. Issues appear in searchable PDF format.

cc-webshop.com









Complete archive includes PDFs of all issues in print through date of purchase.

Input Voltage

Beyond the String

Remember that classic childhood string phone toy? Two cans and a string, a relic of simple communication technology. Yet, it perfectly illustrates the fundamental idea of transmitting sound through vibrations. This journey from string to signal highlights our ongoing quest to share information, a quest that continues to evolve rapidly. Modern telecommunications are profoundly changing how we build interconnected systems. This month's Tech Feature article by Curtis Franklin dives into the topic of telecommunications development.

Delve even further into the complexities of communication with Pedro Bertoleti's piece, "The Edge Computing Paradigm," which explains this distributed computing approach that moves computation and data storage closer to the source of data, improving response times and saving bandwidth compared to solely relying on centralized cloud infrastructure. This local processing is crucial for applications requiring real-time responsiveness and can significantly reduce operational costs by minimizing data transfer to the cloud. Applying edge computing also allows end devices to operate independently, offering resilience even when disconnected.

Building on the concept of processing data closer to the source, Dhairya Parikh presents Part 3 of "Create an AI-Driven Smart Home Security System." This final part of the series enhances a system built on a Raspberry Pi 5 with a Hailo AI Kit by adding face recognition to distinguish between known and unknown individuals, thereby reducing unnecessary alerts. Security is a critical consideration for any connected system, and following the discussion of a smart home security system, Steve Hanna discusses "Zero Trust Architecture for the Internet of Things." This approach moves away from traditional perimeter defenses, asserting that nothing is blindly trusted, and everything must be verified before access is granted. This architecture makes attacks more difficult, allows devices and services to be located anywhere, secures all network traffic, and ensures parties only get access to which they are entitled.

Shifting gears slightly but remaining within the realm of interconnected systems and local control, Braden Trigg, Sebastien Robitaille, Kayleb Stetsko, and Eric Samer share their work on "Fish Sense," a monitoring and control system for aquariums. Designed with modularity and expandability in mind, the system uses ESP32s as microcontrollers for individual devices, communicating over a CAN Bus network using standard Ethernet cables. A central base station, also an ESP32, connects to a web app via the MQTT protocol, allowing users to monitor parameters such as temperature, pH, and water levels, set alarms, and control AC outlets using a custom smart power bar device.

Our regular columnists offer their insights, including Jacob Beningo, explaining how using a meta-tool such as West can revolutionize development by managing complex projects across multiple repositories, enhancing portability and reusability, even if not using the Zephyr RTOS from which it originated. Bob Japenga provides his perspective as he wraps up his series on mobile app development by discussing the process of releasing Flutter apps. Stuart Ball revisits an earlier article to look at calibrating microcontroller clock crystals using GPS technology to improve timing accuracy.

Wrapping up this issue, Bill Li explores "The Future of Graphical User Interfaces," discussing tools such as the Microchip Graphics Suite that streamline GUI development by bridging the gap between UI/UX designers and those implementing the interface on hardware, facilitating a more efficient workflow from design to deployment.

We hope the insights within vibrate clearly from these pages, a sound across a taut string.

KC Prescott

Publisher

kprescott@circuitcellar.com

Issue 419 June 2025 | ISSN 1528-0608

CIRCUIT CELLAR® (ISSN 1528-0608) is published monthly by:

KCK Media Corp.
PO Box 417, Chase City, VA 23924

Periodical rates paid at Chase City, VA, and additional offices. One-year (12 issues) subscription rate US and possessions $50, Canada $65, Foreign/ ROW $75. All subscription orders payable in US funds only via Visa, MasterCard, international postal money order, or check drawn on US bank.

SUBSCRIPTION MANAGEMENT

Online Account Management: circuitcellar.com/account
Renew | Change Address/E-mail | Check Status

CUSTOMER SERVICE

E-mail: customerservice@circuitcellar.com

Phone: 434.533.0246

Mail: Circuit Cellar, PO Box 417, Chase City, VA 23924

Postmaster: Send address changes to
Circuit Cellar, PO Box 417, Chase City, VA 23924

NEW SUBSCRIPTIONS

circuitcellar.com/subscription

ADVERTISING

Contact: Hugh Heinsohn

Phone: 757-525-3677

Fax: 888-980-1303

E-mail: hheinsohn@circuitcellar.com
Advertising rates and terms available on request.

NEW PRODUCTS

E-mail: product-editor@circuitcellar.com

HEAD OFFICE

KCK Media Corp.
PO Box 417
Chase City, VA 23924
Phone: 434-533-0246

COPYRIGHT NOTICE

Entire contents copyright © 2025 by KCK Media Corp. All rights reserved. Circuit Cellar is a registered trademark of KCK Media Corp. Reproduction of this publication in whole or in part without written consent from KCK Media Corp. is prohibited.

DISCLAIMER

KCK Media Corp. makes no warranties and assumes no responsibility or liability of any kind for errors in these programs or schematics or for the consequences of any such errors printed in Circuit Cellar®. Furthermore, because of possible variation in the quality and condition of materials and workmanship of reader-assembled projects, KCK Media Corp. disclaims any responsibility for the safe and proper function of reader-assembled projects based upon or from plans, descriptions, or information published in Circuit Cellar®.

The information provided in Circuit Cellar® by KCK Media Corp. is for educational purposes. KCK Media Corp. makes no claims or warrants that readers have a right to build things based upon these ideas under patent or other relevant intellectual property law in their jurisdiction, or that readers have a right to construct or operate any of the devices described herein under the relevant patent or other intellectual property law of the reader's jurisdiction. The reader assumes any risk of infringement liability for constructing or operating such devices.

© KCK Media Corp. 2025 Printed in the United States

OUR NETWORK

circuit cellar · LinuxGizmos · audioxpress · VOICE COIL · LIS LOUDSPEAKER INDUSTRY SOURCEBOOK

SUPPORTING COMPANIES

Beningo Embedded Group	55
Blackmagic Design	9
CCS, Inc.	61
EmbeddedTS	61, C4
Front Panel	17
Renesas	60
SeaFire Micros, Inc.	C3
Siborg Systems	27
Texas Instruments	29

NOT A SUPPORTING COMPANY YET?

Contact Hugh Heinsohn
hugh@circuitcellar.com,
Phone: 757-525-3677,
Fax: 888-980-1303
to reserve space in the
next issue of *Circuit Cellar.*

THE TEAM

FOUNDER	Steve Ciarcia
PUBLISHER	KC Prescott
CONTROLLER	Chuck Fellows
EDITOR-IN-CHIEF	
CONTRIBUTING EDITOR	Brian Millier
CONTENT MANAGER- EU	Jurij Mikeln
SENIOR ASSOCIATE EDITOR	
	Shannon Becker
TECHNICAL COPY EDITOR	Carol Bower
PROJECT EDITORS	Ken Davidson
	David Tweed
MARKETING MANAGER	Tori Zienka
ADVERTISING COORDINATOR	Diana Winsor
ADVERTISING SALES REP.	Hugh Heinsohn

COLUMNISTS
Stuart Ball (Start to Finish)
Jacob Beningo (Modern Embedded Insights)
Bob Japenga (Embedded in Thin Slices)
Brian Millier (Picking Up Mixed Signals)
Colin O'Flynn (Embedded Systems Essentials)

FEATURES

4 **The Edge Computing Paradigm**
Advantages, Challenges, and Applications in the IoT
By Pedro Bertoleti

10 **Create an AI-Driven Smart Home Security System**
Part 3—Face Recognition and Package Detection
By Dhairya Parikh

18 **Zero Trust Architecture for the Internet of Things**
By Steve Hanna

22 **"Fish Sense"**
A Monitoring and Control System for Aquariums
By Braden Trigg, Sebastien Robitaille, Kayleb Stetsko, and Eric Samer

30 TECHNOLOGY FEATURE
Telecommunications
Bandwidth and So Much More
By Curtis Franklin

36 DATASHEET
AC-DC Power Supplies
More Power — Smaller Footprints
By KC Prescott

COLUMNS

40 Modern Embedded Insights
Improve Portability and Reusability in Embedded Systems with West
By Jacob Beningo

46 Embedded in Thin Slices
An Embedded Designer Looks at Mobile App Development
Releasing Your Flutter App to the App Stores
By Bob Japenga

50 Start to Finish
Crystal Calibration with GPS Redux
By Stuart Ball

62 TECH THE FUTURE
The Future of Graphical User Interfaces
Efficient GUI Design
Experience Flow with Microchip Graphics Suite (MGS)
By Bill Li

57 : PRODUCT NEWS

60 : TEST YOUR EQ

@editor_cc @circuitcellar · circuitcellar · circuitcellar

The Edge Computing Paradigm

Advantages, Challenges, and Applications in the IoT

By
Pedro Bertoleti

In the pre-Internet Dark Ages, all computing was done locally. Then storage and processing moved to the Cloud. Now, local models are gaining traction again. In this article, Pedro explores the fundamentals of edge computing. He explains the edge computing paradigm, compares it to the cloud-centric paradigm, then presents the advantages and challenges of edge computing, and some solutions in the IoT that could benefit from edge computing adoption.

The digital landscape is rapidly evolving, driven by the proliferation of Internet of Things (IoT) devices and the ever-increasing demand for real-time data processing. Traditional cloud computing models, while powerful, often struggle to keep pace with the latency and bandwidth requirements of these emerging applications. Also, from a perspective of cost savings, cloud computing models can be a bad choice—as solutions become more scalable, costs scale as well. The same is true for connectivity—as more data is transferred, the higher costs become.

This is where edge computing steps in, offering a distributed computing paradigm that brings computation and data storage closer to the source of data generation. By processing data at the "edge" of the network—near the devices and sensors that create it—edge computing unlocks a new realm of possibilities for efficiency, responsiveness, and autonomy. These include saving costs from cloud computing usage and connectivity, since only short amounts of data need to be transferred from end devices to the Cloud, processed, and stored in cloud-side.

This article explores the core concepts of edge computing, its advantages and challenges, the typical layers within edge computing architecture, and various solutions that stand to benefit significantly from its adoption. Understanding these fundamentals is crucial for anyone looking to leverage the power of edge computing in today's connected world.

WHAT IS EDGE COMPUTING?

Edge computing, in its essence, is a distributed computing paradigm that brings computation and data storage closer to the location where it is produced and needed—or close to the "edge" of the solution. In lay articles and technical articles and books regarding the Internet of Things (IoT), it improves response times and saves bandwidth. Instead of relying solely on a centralized cloud infrastructure, edge computing leverages devices and systems at the "edge" of the network—near users, sensors, or other data sources—to perform processing tasks locally. This proximity is the key to its benefits.

Edge computing represents a fundamental shift in how we approach data processing and application deployment in the IoT arena. It's not intended to replace the Cloud entirely, but rather to complement it, by offloading certain tasks and enabling new functionalities that are impractical, not affordable, or even impossible with a purely centralized model.

To understand the concept of the edge computing paradigm, a good starting point is to contrast it with the traditional cloud computing paradigm. In a cloud-centric paradigm, data generated by devices is typically transmitted over a network—often the Internet, directly or using gateways to communicate low power wide area networks (LPWANs) and Internet—to a remote data center. There, the data is processed, analyzed, stored, and shown (via dashboards and other methods) to the final users.

Also, data can be sent back to the device. As you may notice, this makes the devices purely terminals, only requiring them to get data from their local environment, and to control some actuator (such as motors or LEDs). This means devices have no sort of intelligence and are nothing but cloud-side interfaces to the field.

This round trip can introduce significant latency, especially when dealing with large volumes of data or applications that require real-time responsiveness, such as devices that deal with IMU data (accelerometer, magnetometer and gyroscope streams of data). Bandwidth limitations can also become a bottleneck in the cloud-centric paradigm, hindering performance.

In addition, the cloud-centric paradigm implies high costs, since cloud-side and connectivity-side costs increase at the same pace (or at a faster pace, in some use cases) as solution scales. In short, the more devices, the more data is generated, and the higher connectivity and cloud costs rise.

The edge computing paradigm addresses these limitations by decentralizing the processing power, relying on processing capabilities of end devices to offload cloud-side and connectivity-side usage. From the last decade on, the processing power offered by cheap microcontrollers and SoCs has grown significantly, allowing embedded devices to process significant amounts of data locally. Then, the edge computing paradigm's main idea is that instead of sending all data to the Cloud, a portion of it is processed locally at the end device (commonly referred to as the "edge device") or in a nearby edge server.

This local processing allows for faster response times, reduced latency, and decreased bandwidth consumption. Only the necessary data or insights are then transmitted to the Cloud for further analysis, long-term storage, or other centralized functions. This makes edge computing a great solution from a technical standpoint, especially for solutions that are intended to scale really high.

From a monetary perspective, the edge computing paradigm works great as well, since it highly reduces cloud-side and connectivity-side costs, resulting in a great and affordable paradigm for solutions that need to scale really high.

The difference between cloud-computing and edge computing paradigms is shown in **Figure 1,** and an example use-case comparison of the two is given below.

Example Use Case Comparison: To get a better idea of edge computing benefits in terms of costs, nothing is better than an analysis of a use case in numbers. Imagine a use case that aims to monitor whether or not the vibration of a piece of equipment is at acceptable levels. This must be done through measurements from a three-axis accelerometer, with each measurement from an axis being a 4-byte float). The analysis occurs on a set of 5,000 accelerometer samples (collected every 10 seconds), totaling a data set to be processed of 60,000 bytes every 10 seconds.

In a traditional scenario (cloud-centric paradigm), every 10 seconds a data set of 60,000 bytes will be transferred, totaling approximately 1.7MB of data transferred per hour, or approximately 1.2GB of data transferred per month, with all the processing of this mass of data occurring entirely in the cloud-side. The cost of this operation would therefore be high, since the connectivity plan must support a monthly traffic of 1.2GB, and the cloud services must be sized to work with this large volume of data.

Now, let's see these numbers when applying the edge computing paradigm to the same use case. To do this, the edge device now will use its processing power, pre-processing these 5,000 samples locally. Instead of sending the 60,000 bytes to the Cloud, it will send only 1 status byte, indicating whether or not the machine in question has acceptable vibration levels. In this scenario with edge computing, the hourly data traffic would drop to only 360 bytes and, consequently, the monthly data traffic would drop to approximately 0.25MB. In other words, with edge computing in this specific solution, the volume of data transferred in the connectivity and processed in the cloud-side is reduced by

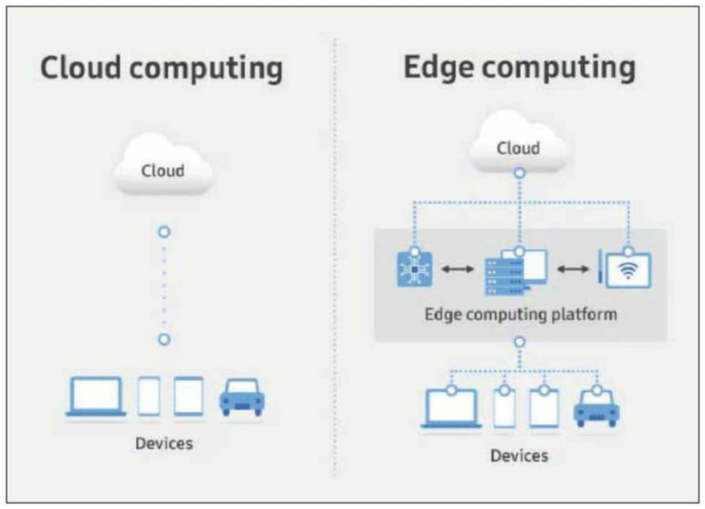


FIGURE 1
Comparison of cloud-computing and edge-computing paradigms. (Source: Samsung [1])

FEATURES

a factor of 5,000, thereby reducing operating costs greatly.

Edge Devices and Infrastructure: The "edge," itself, can encompass a wide range of devices and infrastructure, from the end devices to the Cloud. This infrastructure includes the following elements:

- End devices, meaning devices capable of reading sensors (thermometers, barometers, cameras, etc.) and controlling actuators (motors, LEDs, bulbs, etc.) These devices have significant processing power in their microcontrollers or SoCs, and can perform basic data filtering and pre-processing locally.
- Edge Servers and gateways, commonly called "Fog" (**Figure 2**). Edge servers are dedicated servers deployed closer to the data source (end devices), often located in factories, retail stores, cell towers, or other distributed locations. These servers offer more powerful processing capabilities than individual end devices, and provide a low-latency and low-cost processing power for the end devices. However, due to cost reasons, edge servers aren't commonly used in edge computing paradigm real-world solutions. Gateways are electronic devices that act as intermediaries between the end devices and the Internet/cloud side, providing connectivity, security, and basic data aggregation.
- Cloud server and services, which process, analyze, show, and store the data sent by the end devices.

Figure 2 shows the constituents of the most common edge computing infrastructure, and also shows the orders of magnitude between the infrastructure elements.

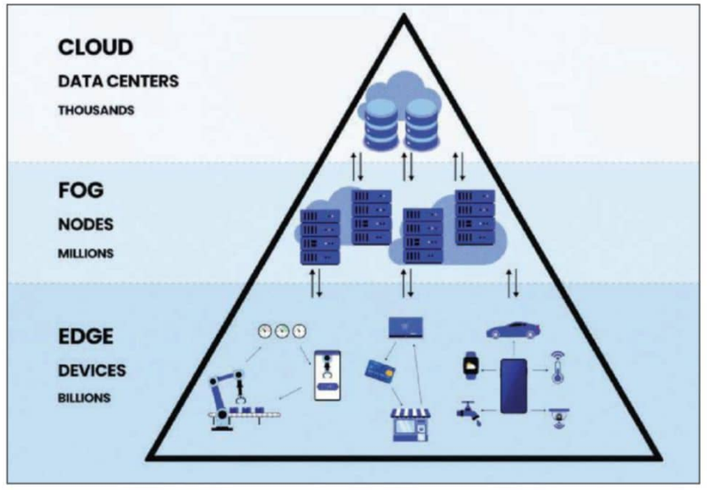


FIGURE 2
The most common edge computing paradigm infrastructure. (Source: Macrometa [2])

The edge computing paradigm is also successfully used for some connectivity models, such as long range wide area network (LoRaWAN). LoRaWAN is a LPWAN connectivity, mainly designed for scalable solutions, excelling in long-range communication, low bandwidth, and low power consumption. The LoRaWAN infrastructure (shown in **Figure 3**) is an example of the edge computing paradigm, since it has the three elements of an edge computing paradigm solution: end devices (capable of reading sensors, processing its data, and transferring only small data to gateways and servers); gateways and edge servers (represented here by Network Server and Application Server); and finally the Cloud services (cloud-side).

SPECIAL NEEDS OF AN EDGE COMPUTING PARADIGM END DEVICE

The end devices that are part of a solution using the edge computing paradigm have some special needs in terms of software development.

As discussed earlier, the end devices in edge computing play a major role in the whole solution, since they process the raw data, and all layers above them (Fog and cloud-side layers) assume this raw data processing is correct. Then, because the responsibility of end devices is high, the embedded software quality must also be as high as possible. One of the best practices to ensure good embedded software "quality"—meaning stable and reliable—is to use a well-known and well-tested embedded operating system.

There are basically two operating system options for embedded software developers to use:

- Real-Time Operating System (RTOS) prioritizes predictable timing (determinism) and low latency for tasks with strict deadlines, often used in resource-constrained devices for control systems. Examples of well-known and well-tested RTOSes are FreeRTOS, Zephyr, and NuttX.
- General Purpose Operating System (GPOS) prioritizes versatility, features, and resource sharing for diverse applications, offering a richer environment but typically lacking strict real-time guarantees. Examples of well-known and well-tested GPOSes are embedded Linux distros and Android.

Additionally, because end devices need to work flawlessly 24 hours a day and 7 days a week, it's mandatory that their embedded software has good quality in testing and software integration as well. Then, applying DevOps concepts (including unitary tests, continuous delivery, and continuous integration) is a good call, since it ensures tests and development occur as safely and smoothly as possible, even when a team of several developers is working on the same solution at the same time.

KEY CHARACTERISTICS AND MAIN ADVANTAGES OF EDGE COMPUTING

By now, you should be familiar with what the edge computing paradigm stands for and why it's noteworthy for IoT solutions. Now, it's time for a deeper dive into the advantages and key characteristics that edge computing offers.

1) Processing data closer to the source reduces latency and improves response times. It means devices can take much faster actions, not requiring cloud-side to decide what and when must be done in the end device. It enables real-time or near-real-time data processing and analysis, which is critical for applications that require immediate feedback.
2) By distributing the computation and storage across multiple edge devices and servers, the connectivity and cloud will be offloaded, resulting in reduced operational costs. Also, since the processing doesn't rely on only one element (cloud-side), the solution general confidence also increases.
3) As the edge computing paradigm implies, in transferring to the Cloud only essential data (results of locally pre-processed data in end devices), very cheap, long-range and bandwidth connectivity can be applied, such as LoRaWAN and SigFox.
4) End devices can operate independently, even when disconnected from the Cloud, providing resilience and fault tolerance.
5) A solution that applies the edge computing paradigm can be easily scalable, with connectivity and cloud costs increasing at a slower pace.
6) Processing sensitive data locally significantly reduces the risk of data breaches and privacy violations. Data can be anonymized or aggregated at the edge-side before being transmitted to the cloud-side.
7) By distributing processing power across multiple edge devices, edge computing increases the reliability and resilience of the system. If one edge device fails, the others can continue to operate.
8) Advances in embedded systems now allow for deployment of lightweight AI models directly on edge devices (also known as TinyML). This enables real-time inference and decision-making without needing cloud processing. Examples include predictive maintenance in industrial settings or personalized recommendations in retail.

DRAWBACKS OF EDGE COMPUTING

Clearly, edge computing offers numerous advantages. However, like any paradigm, it also presents several challenges and drawbacks that must be carefully considered when adopting it to a real scalable solution. The five main points of attention in the edge computing paradigm are discussed below.

First, end devices of an edge computing solution need to process data locally. It means these devices often require more robust hardware with greater processing power and storage capacity, leading to higher upfront costs when compared to end devices used in cloud-centric paradigm solutions.

Second, while the distributed nature of the edge computing paradigm makes it harder to get at sensitive data, it also exposes a new attack surface—hardware hacking. If a hacker has access to the end device's hardware, it configures a security threat. Therefore, physical measures to increase hardware security of edge devices are crucial.

Third, developing and deploying applications for solutions that rely on edge computing can be more complex than traditional cloud-centric solutions development, since end devices must handle more complex and critical data processing, compared to cloud-centric solutions. Therefore, the efforts in embedded software development and hardware development are usually high.

Fourth, end devices of a solution that applies the edge computing paradigm tend to have high power consumption, since they must process data locally. It means these devices aren't commonly low-power devices.

Fifth, even though edge devices have significant processing power, they still have well-defined limitations, especially compared to the processing power of cloud servers. This requires careful optimization of applications and algorithms that run in the end device's side.

ABOUT THE AUTHOR

Pedro Bertoleti is a Brazilian embedded systems engineer and publisher on a Brazilian embedded systems website called Embarcados (www.embarcados. com.br). He has strong experience in development with PIC, ESP32, ARM Cortex M0 and M3, and Rabbit RCM22xx microcontrollers. Pedro also has strong knowledge and experience in IoT projects and Embedded Linux Embedded Systems development.

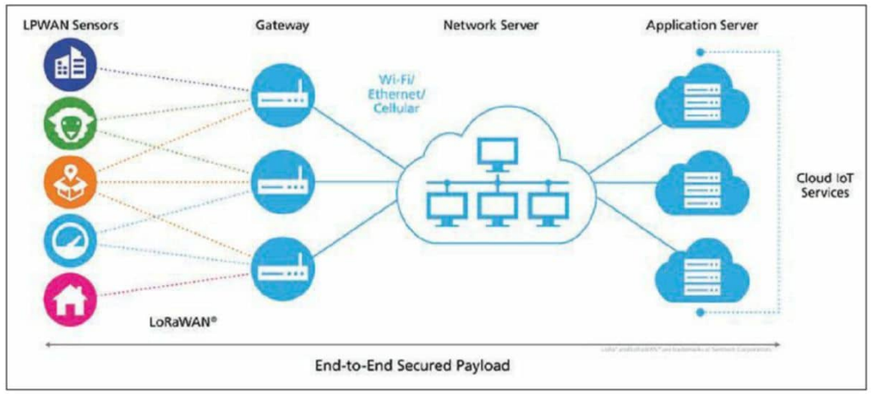


FIGURE 3
Edge computing paradigm applied to LoRaWAN infrastructure. (Source: Semtec [3])

COMMON SOLUTIONS THAT BENEFIT FROM EDGE COMPUTING

Edge computing is transforming a wide range of industries and applications, since real scalable solutions can grow at affordable costs in terms of connectivity and cloud usage. Some examples of the most common solutions that benefit from edge computing are given here. As the cost of edge devices decreases and the technology matures, we can expect to see even wider adoption of edge computing in the years to come.

Autonomous Vehicles: Existing and upcoming self-driving cars strongly rely on real-time data processing to make critical decisions as fast as possible, especially because connectivity and cloud sites present significant latency, and connectivity isn't available everywhere and every time it's needed. Edge computing in this use case enables vehicles to process sensor data locally, reducing latency and improving safety, and only transfer and receive data to or from the Cloud when necessary.

Industrial Automation: In manufacturing environments, one of the most valuable pieces of advice is, "Better safe than sorry." Edge computing end devices can monitor equipment performance, detect anomalies (using vibration and temperature measurements, for example), and indicate emerging problems. This can help indicate when the predictive maintenance needs to take action, and consequently reduces industrial equipment downtime.

Healthcare: Edge computing end devices can be used to monitor patients remotely, processing all vital signs and providing a patient's status to the cloud-side solution. Also, using TinyML and AI-powered features, these end devices can foresee some health problems, thereby enabling timely interventions by providers and better patient care and outcomes.

Smart Cities: Edge computing can be used to improve the efficiency and sustainability of urban environments. Examples include smart streetlights that adjust brightness based on traffic patterns, environmental sensors that monitor air quality, and smart parking systems that optimize parking availability.

CONCLUSION

"Edge computing" is more than just a buzzword or a tech trend. It's a fundamental shift in how we think about data processing and application deployment in IoT solutions. By bringing computation closer to the source of data, the edge computing paradigm unlocks a new wave of innovation, empowering real-time applications, reducing bandwidth costs, and enhancing security and autonomy. While challenges remain in terms of complexity, hardware security, and local processing management by end devices' embedded systems, the potential benefits of edge computing are undeniable.

As the number of connected devices continues to grow, the demand for edge computing solutions tends to increase. By understanding the core concepts, advantages, and challenges of the edge computing paradigm, businesses and developers can leverage its power to create innovative, high-impact solutions that transform industries and improve lives. Embracing edge computing is no longer a futuristic concept—it's a strategic imperative for success in the increasingly connected world.

Additional materials from the author are available at:
www.circuitcellar.com/article-materials

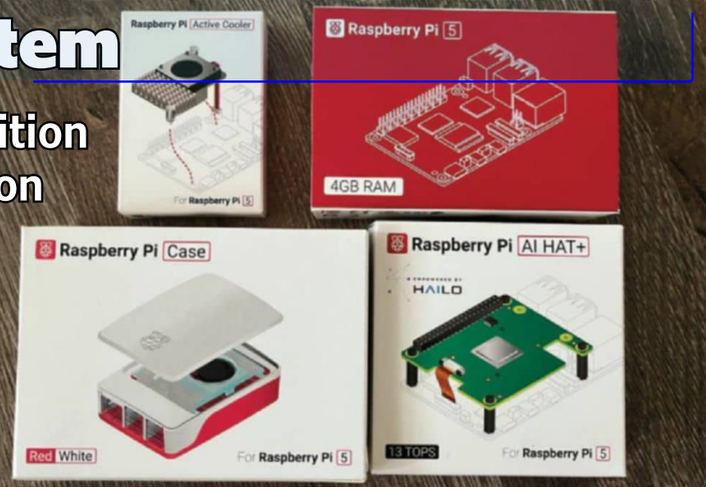
References [1] to [3] as marked in the article can be found there.

FREE Download Now!
Mac, Windows and Linux
Blackmagicdesign
Edit and color correct using the same software used by Hollywood, for free!
DaVinci Resolve is Hollywood's most popular software! Now it's easy to create feature film quality videos by using professional color correction, editing, audio and visual effects. Because DaVinci Resolve is free, you're not locked into a cloud license so you won't lose your work if you stop paying a monthly fee. There's no monthly fee, no embedded ads and no user tracking.
Editing, Color, Audio and Effects!
DaVinci Resolve is the world's only solution that combines editing, color correction, visual effects, motion graphics and audio post production all in one software tool! You can work faster because you don't have to learn multiple apps or switch software for different tasks. For example, just click the color page for color, or the edit page for editing! It's so incredibly fast!
Professional Editing
DaVinci Resolve is perfect for editing sales or training videos! The familiar track layout makes it easy to learn, while being powerful enough for professional editors. You also get a library full of hundreds of titles, transitions and effects that you can add and animate! Plus, DaVinci Resolve is used on high end work, so you are learning advanced skills used in TV and film.
Creative Color Correction
DaVinci Resolve's color page is Hollywood's most advanced color corrector and has been used on more feature films and television shows than any other system! It has exciting new features to make it easier to get amazing results, even while learning the more advanced color correction tools. There's PowerWindows™, qualifiers, tracking, advanced HDR grading tools and more!
Designed to Grow With You
DaVinci Resolve is designed for collaboration so as you work on larger jobs you can add users and all work on the same projects, at the same time. You can also expand DaVinci Resolve by adding a range of color control panels that let you create unique looks that are impossible with a mouse and keyboard. There's also edit keyboards and Fairlight audio consoles for sound studios!
DaVinci Resolve 19 ..... Free
DaVinci Resolve Micro Color Panel ..... Only $559
Learn the basics for free then get more creative control with our accessories!
Learn more at www.blackmagicdesign.com
Download free on the DaVinci Resolve website
NO SUBSCRIPTIONS • NO ADS • NO USER TRACKING • NO AI TRAINING

FEATURES

Create an AI-Driven Smart Home Security System

Part 3—Face Recognition and Package Detection



By
Dhairya Parikh

In the pre-Internet Dark Ages, all computing was done locally. Then storage and processing moved to the Cloud. Now, local models are gaining traction again. In this article, Pedro explores the fundamentals of edge computing. He explains the edge computing paradigm, compares it to the cloud-centric paradigm, then presents the advantages and challenges of edge computing, and some solutions in the IoT that could benefit from edge computing adoption.

In Parts 1 and 2 of this article series in the March and April, 2025 issues of Circuit Cellar [1] [2], we started building our AI-powered home security system with a Raspberry Pi 5 and a Hailo AI Kit. We gave it three key functionalities: The ability to detect people in real-time using YOLO (You Only Look Once) models running on the Hailo AI Kit. The model was configured to identify people with high confidence (which is customizable) and reduce false alarms. With an email alert system that notified us whenever an intruder was detected, using a Gmail account with two-step verification and app passwords. We can monitor the camera feed from any device on the same network and view detection results in real time via a local server using the Flask framework in Python to stream the security feed from the Raspberry Pi.

IMPROVEMENTS

Our home security system is already capable of many things, such as detecting intruders, sending timely alerts, and providing real-time video feeds via a web interface (**Figure 1**). Although the current implementation is functional and adequate in some scenarios, a few gaps still need to be addressed to make the system truly reliable and user friendly.

Let's look at some of these limitations in more detail, to get a better idea of where our system falls short:

Lack of Advanced Person Detection: While the current system can detect people, it cannot detect who the person is. Just imagine a scenario where you install this in your home, and you are alerted every time a family member is detected in the frame. This information won't be much use, since family members are not a threat to our home's security. Hence, we need a way for the system to distinguish between known and unknown identities.

Lack of Advanced Features: Although we currently have a feature-rich security system, we can implement a lot of potential improvements to make it even better. For instance, we can add an ability to detect when packages are delivered to our doorstep (no one knocks or hand delivers them), so that we can avoid "porch pirates" and exposure to bad weather in certain scenarios.

To address these limitations, we will now add two advanced features—face recognition and package detection.

Face Recognition: By integrating face recognition, the system will be able to identify family members and friends and reduce unnecessary alerts. This feature can be trained to recognize family members, friends, and frequent visitors, allowing the system to notify only when an unrecognized person is detected.

Package Detection: Adding an object detection model to recognize packages will enable the system to notify users when a delivery is made. While the YOLO model does not support this directly, I'll create a mock scenario for you, and I will provide you with a great starter guide to implement this feature.

With these improvements, our AI-driven home security system will move one step closer to becoming an intelligent and context-aware security solution that enhances both convenience and safety. Let's get started by exploring how we can integrate these features into our system.

IMPROVING OUR AI-POWERED HOME SECURITY SYSTEM

The first step was to connect all the hardware together and complete all the steps given in Part 2 of this series [2]. Make sure that you have a running security system before you take the next steps. What this essentially means is that you should be able to see your camera's output on the web interface, as shown in Figure 1.

Similar to what we did in Part 2, let's get an idea of how we will be implementing these features into our system before starting with the actual code development.

Face Recognition: By applying this additional filter, we ensure that alerts are triggered only when unknown individuals are detected, reducing false alarms and enhancing system reliability. To integrate face recognition, we will leverage the popular Python package, "face_recognition," which simplifies the process by allowing us to recognize faces with just a single reference image. (Don't worry, I'll cover how to install this.) The flow will be as described below and shown in **Figure 2**:

- ***Load Known Faces:*** We will load images of family members or frequent visitors and encode them for easy comparison. These face encodings will be stored in memory for quick access.
- ***Face Extraction and Comparison:*** During the detection process, each frame from the camera feed will be analyzed to extract any detected faces. These faces will then be compared against the known face encodings to check for a match.
- ***Decision to Send Alert:*** If the detected face is recognized, the system will suppress the alert, preventing unnecessary notifications. However, if an unrecognized face is detected, an alert will be sent to notify the user.

Package Detection (Simulation): YOLO models, while excellent at detecting objects, do not have a dedicated label for packages or deliveries. To demonstrate package detection,

FIGURE 1
Web interface we created in the Part 2 of this series. (Yes, that is me!). The stream and detection updates are available across the local network.

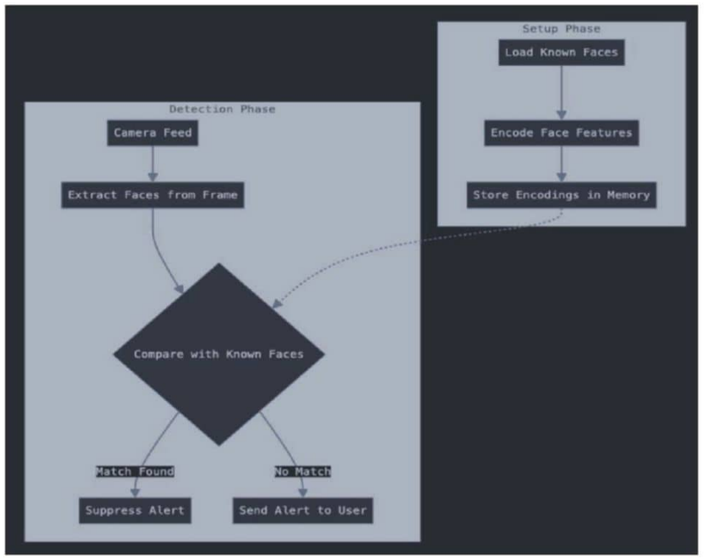


FIGURE 2
Flow diagram explaining how the Face Recognition implementation will work in our security system. It involves an initial set-up phase and frame by frame processing.

FEATURES

```
cd hailo-rpi5-examples
source setup_env.sh
git clone <repository-link> # available in the references and
README file
cd <repository-name>
pip install -r requirements.txt
```

LISTING 1
Commands to run on a terminal window to install all the additional dependencies for this project.

we will use a creative workaround: We will treat the detection of a bottle as a proxy for identifying a package. While this may not be a perfect real-world analogy, it will allow us to showcase the full flow of adding object detection and sending alerts.

When a bottle is detected in the frame, we will enclose it within a distinct colored bounding box (blue) to differentiate it from person detections. Once a package is "detected," a separate alert will be sent to the user, notifying them of a potential delivery. We will also enhance our existing Flask interface to display package detection information alongside intruder alerts, thus providing a unified view of all security events.

To implement these new features, we will make modifications to the core detection code (home_security_system_core.py) and the streaming interface (stream_security_video.py). You can find both of these files in the code repository for Part 2 of this article series [3]. Both face recognition and package detection will be seamlessly integrated into the existing system, to ensure that all detections, whether person or package, are handled perfectly.

Now that we have a solid foundation for the changes, we are now ready to dive into the implementation of face recognition and package detection in our AI-powered security system. Let's get our hands dirty and start updating the code!

CODE WALK-THROUGH

We will be walking through all the existing files from Part 2 of this series that need to be updated to add the two new features to our system. All the code discussed here is available in the Part 3 dedicated GitHub repository [4]. The files that will be updated are:

- `home_security_system_core.py`: The main file in which we will implement the face recognition and package detection features.
- `stream_security_video.py`: The file that handles the streaming part.
- `index.html`: The file that handles the interface we show for streaming our camera.

You can use the Part 2 repository as a starting point and build from there. For this article, we will focus mainly on the home_security_system_core.py, since this is where the main logic changes are needed.

HOME SECURITY SYSTEM CORE CODE

We will start by adding three packages that we will be using for face recognition: NumPy (Python's scientific computing package); face_recognition [5]; dlib (dependency for the face_recognition package) [6]; and the repository of Hailo examples for the Raspberry Pi 5 [4]. To make things easier for you, I have added a requirements.txt file in this article's code repository [4]. **Listing 1** contains the commands to run on a terminal window to install all the additional dependencies for this project.

After installing the packages, we are ready to start making code changes. Let's start by importing the two new libraries that we will be using.

```
import face_recognition
  # Added for face recognition
import numpy as np
  # Added for face recognition operations
```

Next, we add functions to the `IntruderDetectionCallback` class we created in Part 2, and update a few functions as well. Let's start by creating two new functions: `load_known_faces` and `recognize_faces` (**Listing 2**).

The first function will be used to load the known face images. For this, you will have to add the images of you and your family members in the "known_faces" folder of your repository (it will be empty by default). Make sure that you rename the image file to the name of the person who is in the image, since that is what the system uses to identify the person.

```
self.last_package_alert_time = datetime.min # Added for package alerts

# Face recognition initialization - NEW
 self.known_face_dir = "./known_faces" # Directory with known face images
 self.known_face_encodings = []
 self.known_face_names = []
 self.load_known_faces()
```

LISTING 3
Additions required in the `__init__` function to execute the loading of all the known faces and initializing certain variables we would need.

```
def load_known_faces(self):
 """Load known faces from the specified directory"""
 logger.info(f"Loading known faces from {self.known_face_dir}")

 try:
 if not os.path.exists(self.known_face_dir):
 logger.warning(
  f"Known faces directory '{self.known_face_dir}' does not exist"
 )
 return

 for filename in os.listdir(self.known_face_dir):
 if filename.endswith((".jpg", ".jpeg", ".png")):
  name = os.path.splitext(filename)[0] # Get the name from the filename
  image_path = os.path.join(self.known_face_dir, filename)

  try:
  face_image = face_recognition.load_image_file(image_path)
  face_encodings = face_recognition.face_encodings(face_image)

  if face_encodings:
  self.known_face_encodings.append(face_encodings[0])
  self.known_face_names.append(name)
  logger.info(f"Loaded face for {name}")
  else:
  logger.warning(f"No face found in {filename}")

  except Exception as e:
  logger.error(f"Error loading face from {filename}: {str(e)}")

 logger.info(f"Loaded {len(self.known_face_names)} known faces")
 except Exception as e:
 logger.error(f"Error during face loading: {str(e)}")
```

LISTING 2
We add new functions to the `IntruderDetectionCallback` class we created in Part 2, `load_known_faces` and `recognize_faces`.

```
def recognize_faces(self, frame):
  """Recognize faces in the frame and return list of known faces found"""
  recognized_faces = []
  try:
   if not self.known_face_encodings:
   return recognized_faces

  # Resize frame for faster face recognition
  small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

  # Convert the image from BGR color (OpenCV) to RGB color (face_recognition)
  rgb_small_frame = cv2.cvtColor(small_frame, cv2.COLOR_BGR2RGB)

  # Find all face locations and face encodings in the current frame
  face_locations = face_recognition.face_locations(rgb_small_frame)
  face_encodings = face_recognition.face_encodings(
   rgb_small_frame, face_locations
  )

  for face_encoding, face_location in zip(face_encodings, face_locations):
   # See if the face is a match for the known face(s)
   matches = face_recognition.compare_faces(
    self.known_face_encodings, face_encoding
   )
   name = "Unknown"

   # Use the known face with the smallest distance to the new face
   face_distances = face_recognition.face_distance(
    self.known_face_encodings, face_encoding
   )
   if len(face_distances) > 0:
    best_match_index = np.argmin(face_distances)
    if matches[best_match_index]:
     name = self.known_face_names[best_match_index]
     recognized_faces.append(name)
 except Exception as e:
  logger.error(f"Error during face recognition: {str(e)}")

 return recognized_faces
```

LISTING 4
The core function that implements the face recognition functionality. It compares the detected faces with all the known faces, and if someone matches, it returns the person's name.

FEATURES

```
        # Process each detection
        for detection in detections:
          try:
            confidence = detection.get_confidence()
            label = detection.get_label()

            if confidence > user_data.confidence_threshold:
              bbox = detection.get_bbox()
              track = detection.get_objects_typed(hailo.HAILO_UNIQUE_ID)
              track_id = track[0].get_id() if track else 0

              # Person detection
              if label == "person":
                # Only add to detection info if not a known face
                if not recognized_faces:
                  person_detection_info.append(
                    {"id": track_id, "type": "person", "confidence": confidence}
                  )

                # Draw bounding box if frame is available
                if frame is not None:
                  x1, y1, x2, y2 = map(
                    int,
                    [
                      bbox.xmin() * width,
                      bbox.ymin() * height,
                      bbox.xmax() * width,
                      bbox.ymax() * height,
                    ],
                  )
                  # Use green for known persons, red for unknown persons (intruders)
                  color = (0, 255, 0) if recognized_faces else (0, 0, 255)
                  cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)

                  label_text = f"ID: {track_id}"
                  if recognized_faces:
                    label_text = f"ID: {track_id} - Known: {', '.join(recognized_faces)}"

                  cv2.putText(
                    frame,
                    label_text,
                    (x1, y1 - 10),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    0.5,
                    color,
                    2,
                  )

            # Package detection (using bottle as proxy)
            elif label == "bottle":  # Using bottle detection as package proxy
              package_detection_info.append(
                {"id": track_id, "type": "package", "confidence": confidence}
              )

              # Draw bounding box if frame is available
              if frame is not None:
                x1, y1, x2, y2 = map(
                  int,
                  [
                    bbox.xmin() * width,
                    bbox.ymin() * height,
                    bbox.xmax() * width,
                    bbox.ymax() * height,
                  ],
                )
                # Use blue for packages
                cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 0), 2)
                cv2.putText(
                  frame,
                  f"Package {track_id}",
                  (x1, y1 - 10),
                  cv2.FONT_HERSHEY_SIMPLEX,
                  0.5,
                  (255, 0, 0),
                  2,
                )
          except Exception as e:
            logger.error(f"Error processing detection: {str(e)}")
            continue
```

LISTING 5
(Continued)

Now, we will call this function when we start Security system. For this, we will have to make a few changes in the `__init__` function (**Listing 3**).

Note that apart from calling the newly created function, we also define a few new variables that we will be using later in our code. For instance, we use the variable `last_package_alert_time` to track the last arrived package and the face encoding and names to keep a track of all the known faces in our directory along with their identities.

Next, let's define the `recognize_faces` function, which is the main logic behind how faces are recognized in each frame (**Listing 4**).

In this function, we do the following for each frame:

- Transform the frame to make it smaller, and convert it into RGB color mode.
- Identify where the face is in the frame, and fetch it.
- Create encodings from the face image, and compare it with all the known encodings.
- If the comparison yields a positive result, add the identity (name) of that person to the recognized faces list.
- Return this list.

We have now successfully added face recognition to our system! In addition to the code in Listing 4, I have added a few more things to get cleaner logs, so that you don't get overwhelmed by them. I am not covering those changes here because they aren't as important.

Additionally, I have made some changes in the existing email alert functions to send package detection alerts. Since we covered this code in Part 2, I am not discussing it here. You can check the code in the repository [4], since I have added comments wherever necessary for easier understanding.

Our work is not done yet! We still must write the logic to run the face recognition function and package detection logic for every camera frame we get. For that, we must make changes in the `app_callback` function (the code that runs for each frame).

To enhance the functionality of our AI-driven security system, we have made several significant changes to the `app_callback` function. These additions allow us to incorporate face recognition for identifying known individuals and package detection using a YOLO proxy label. The final updated function is given in **Listing 5.**

```
2025-03-20 19:19:05,045 - INFO - Configuration successful!
Picamera2 configuration: width=1280, height=720, format=RGB
2025-03-20 19:19:05,168 - INFO - Camera started
picamera_process started
2025-03-20 19:19:07,154 - INFO - Email alert sent: intruder
2025-03-20 19:19:07,209 - INFO - Detection data sent to server: intruder
2025-03-20 19:19:07,209 - INFO - Intruder alert email sent and detection data updated
2025-03-20 19:19:07,289 - INFO - Skipping intruder alert due to cooldown period
2025-03-20 19:19:17,931 - INFO - Known person detected: dhairya
2025-03-20 19:19:31,106 - INFO - Email alert sent: package
2025-03-20 19:19:31,161 - INFO - Detection data sent to server: package
2025-03-20 19:19:31,161 - INFO - Package alert email sent and detection data updated
2025-03-20 19:19:31,226 - INFO - Skipping package alert due to cooldown period
```

FIGURE 3
Updated logs showing the system's capability to detect known people, intruders, and packages (simulated as a bottle for now). Additional logs are printed for more information.

LISTING 5
The final updated callback function with face recognition and package detection features implemented. This is the function that runs for each captured frame of our system.

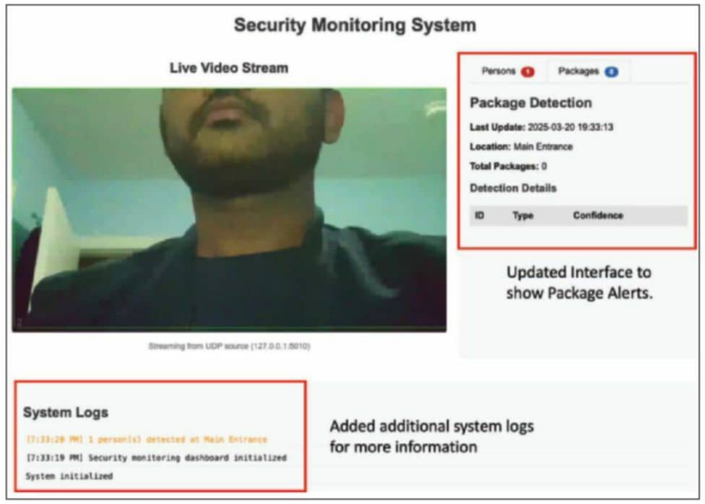


FIGURE 4
The updated user interface for our home security system. Major additions include a new tab for package updates and system logs on the interface, itself.

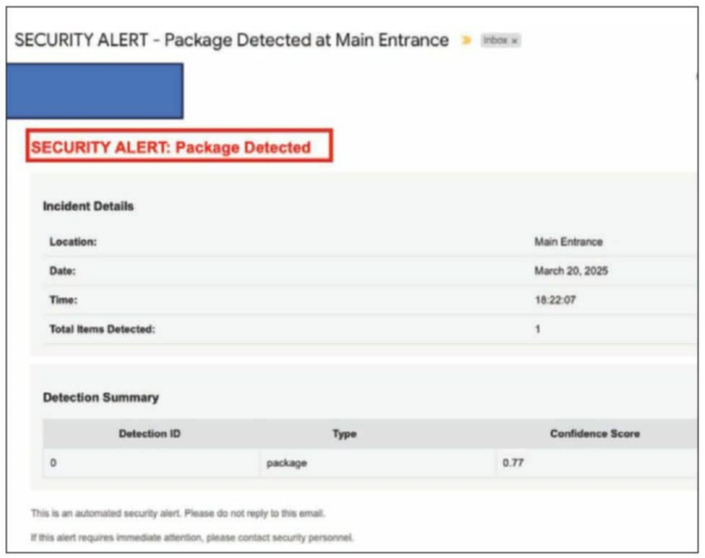


FIGURE 5
A screenshot of the email that will be sent from our home security system every time a new package is detected. Note that the structure and format of the email is similar to the intruder detection alerts, but customized for packages.

Additional materials from the author are available at:
www.circuitcellar.com/article-materials
References [1] to [6] as marked in the article can be found there.

RESOURCES

Raspberry Pi | www.raspberrypi.com

Python | www.python.org

KEY CODE ADDITIONS TO IMPLEMENT NEW FEATURES

The actions of the key additions to the project's code that were made to implement the proposed new features are summarized here.

Face Recognition Integration:

- Recognize Faces in Each Frame: Before processing object detections, the updated code now extracts and processes each frame for face recognition. The function `user_data.recognize_faces(frame)` compares detected faces with known encodings.
- Filter Out Recognized Faces: If a recognized face is found, it suppresses unnecessary alerts by preventing the system from treating that detection as an intruder. Only unknown faces trigger intruder alerts.
- Visual Indicators for Recognized/Unknown Faces: A bounding box with different colors is drawn around detected faces—green for recognized faces and red for unknown intruders. The label also updates dynamically to show the names of recognized individuals.

Package Detection Using YOLO Proxy:

- Simulated Package Detection: Since YOLO does not have a predefined label for packages, we are using a "bottle" as a proxy to demonstrate package detection. When a bottle is detected, it is treated as a package.
- Drawing Bounding Boxes with Custom Color: Detected packages (bottles) are enclosed within a blue bounding box to distinguish them from green and red bounding boxes used for known person and intruder detections.
- Separate Package Alerts: Upon package detection, a separate alert is sent to the user, notifying them of a potential delivery.

THE SYSTEM IN ACTION

That's it for this file. We have successfully made all the changes that were needed. We are now ready to see our system in action. You can run this updated file and check out the logs by bringing an unknown person or a bottle into the frame to see the new additions in action. Moreover, if you have your own face in the system and you enter the frame, you will see a message saying that a known person was detected. Cool, right? Example logs are shown in **Figure 3.**

In addition to the listed changes, I have altered a few things for the streaming files (the HTML file and the backend Python file stream_security_video.py) to include the package detection updates on the new interface. These code changes are pretty simple and self-explanatory, so I am not describing them here. You can review the new files in the code repository.

Once you have made all the changes, the updated interface should show package updates in a new tab. Refer to **Figure 4** to get a better look at the refreshed interface. There is also a new email alert for packages. You can see a snapshot of what this alert will look like in **Figure 5**.

CONCLUSION AND NEXT STEPS

That's it for this three-part article series! We have successfully built our very own AI-powered security system with a Raspberry Pi, a Hailo AI Kit, and a Raspberry Pi camera or other supported USB camera.

In Part 1, we set up the Raspberry Pi and ran our first AI model. In Part 2, we added person detection, email alerts, and a local web interface to monitor the security feed. And in this final part of the series, we enhanced the system by integrating face recognition to identify known individuals, and simulating package detection to notify users about potential deliveries. The result is a robust and intelligent security system that can effectively monitor your home and keep you informed in real time.

While this marks the end of this series, there are still several ways to take this project even further. You can add real-time mobile notifications using push services, integrate cloud storage to log detection data and video feeds, or even scale the system to handle multiple cameras for greater coverage. Additionally, advanced AI models can be used for behavior analysis to detect suspicious activities and improve overall system accuracy.

If you're looking for more project ideas, consider building a smart doorbell with face recognition, developing an AI-powered pet monitor to keep an eye on your furry friends, or creating an automated package locker system that secures deliveries and notifies you instantly. The possibilities are endless, and with the power of AI and IoT, you can continue to create innovative and high-impact projects.

ABOUT THE AUTHOR

Dhairya Parikh is a seasoned Technical Writer with more than five years of expertise in IoT, Machine Learning, and Hardware. He has authored a book on Raspberry Pi, published numerous technical articles, and currently contributes to RaspberryTips, a leading Raspberry Pi blog. With a talent for simplifying complex topics, Dhairya creates engaging step-by-step guides for developers and tech enthusiasts. He has completed his master's in computer engineering from the University of Waterloo, specializing in AI and Machine Learning.

Zero Trust Architecture for the Internet of Things

By
Steve Hanna,
Infineon Technologies

Discover how Zero Trust Architecture is transforming IoT security, moving beyond outdated perimeter defenses to ensure every device, user, and service is authenticated and secure. Learn why adopting ZTA is essential for modern IoT design and how it can boost security, flexibility, and trust in your connected systems.

Traditional cybersecurity approaches focus on creating a trusted network of trusted devices and services with a perimeter of firewalls to keep attackers out. However, recent changes are forcing that "perimeter security architecture" to be abandoned for a more sophisticated "Zero Trust Architecture" (ZTA) where devices and services can reside anywhere globally, and no device or service is completely trusted.

A similar change is now coming to the Internet of Things (IoT), even in the Smart Home. This white paper explains how IoT security is being transformed by Zero Trust Architecture, why this change is happening, and what IoT designers must do to adapt.

PROBLEMS WITH PERIMETER SECURITY

For many years, cybersecurity used perimeter-based security with firewalls. This approach harkens back to the time of castles and moats in the Middle Ages. By keeping the bad guys outside of a castle's walls with a moat, the good (trusted) guys inside thought they were safe. When inside, no extra protection, such as armor, was necessary. How easy and efficient!

Unfortunately, perimeter security does not work well in modern cybersecurity. Attackers can use an email with a malicious link or a tempting ad on a web page to infect machines inside the perimeter. An infected machine becomes a base for observing unencrypted network traffic, scouting the target network, and infecting other machines. A more fundamental problem with the perimeter model is that users today spend much of their time outside the office and trusted services are often located in external clouds. Accommodating remote work in a perimeter security model requires extending the perimeter to include remote users and cloud services, which is a tremendous extension—and a substantial risk.

Even in those workplaces where everyone and everything is in the office, perimeter-based security means that it only takes one infected device or malicious user to cause a network compromise. This has always been a problem.

ZERO TRUST ARCHITECTURE: THE MODERN CYBER SECURITY APPROACH

Over the last 20 years, a much better approach for cybersecurity has been developed: Zero Trust Architecture (ZTA). With ZTA, nothing is blindly trusted. Everything must be verified before it can be trusted.

Here is a brief summary of the principles of ZTA [1]:

1. No network is trusted so all communications are secured.
2. All parties (users, services, devices, etc.) are authenticated.
3. Dynamic policies determine which access requests are authorized.
4. The security of users, networks, devices, software, and data is monitored.
5. When a problem is detected, it is addressed promptly.

These ZTA principles are simple, but they are revolutionary and have many advantages:

- Successful attacks are more difficult because easy targets are eliminated.
- Users, devices, and services can be located at any location desired.
- Any network can be used because all network traffic is secured.
- Each user, device, and service gets only the access to which it is entitled.
- Unauthorized parties get no access.
- If an attacker does compromise a user, network, device, or service, the compromise can be detected quickly leading to rapid containment and response.

The main challenges of ZTA are:

- Legacy systems are hard to change.
- Additional processing power is needed to encrypt and authenticate all messages.
- Monitoring the security of networks, devices, and software requires automation to scale.

Despite these challenges, the ZTA principles have been developed, tested, adopted, and refined over the last several decades across academia, the private sector, and the public sector. They have proven effective at reducing the impact of security compromises while increasing flexibility. And they continue to be refined.

Now ZTA is moving into widespread adoption. Government agencies have been especially vocal in advocating the use of ZTA. Back in 2019, the UK's National Cyber Security Centre (NCSC) laid out the principles of ZTA and recommended adoption. In 2020, the US National Institute for Standards and Technology (NIST) further defined ZTA and detailed several options for ZTA deployment, including variations that permit ZTA to be overlaid onto legacy systems that do not support ZTA natively. For example, ZTA can be implemented in software agents and hardware gateways that protect legacy resources.

With the announcement on May 12, 2021 of Executive Order (EO) 14028 (Improving the Nation's Cybersecurity), US Government Federal agencies were tasked with implementing a ZTA approach to security [2]. Every Federal agency is required to immediately develop a plan and start to transition to ZTA.

Across the private sector, ZTA adoption is also growing. A recent survey [3] shows that more than 86% of companies have started to implement ZTA. However, only 2% have fully implemented all of the ZTA tenets. Results so far are promising. Companies that have implemented all ZTA tenets see half the incidents as compared to those that have implemented none.

ZERO TRUST ARCHITECTURE FOR EMBEDDED SYSTEMS

When building embedded systems, security has often been an afterthought. System designers hoped that nobody would bother to hack embedded systems and they worried that including proper security would add cost and complexity.

However, several new trends are leading to the conclusion that building in good security from the start actually reduces cost and complexity while increasing likely sales.

Low-power processors such as the Infineon PSOC 4, PSOC 6, and PSOC Edge microcontroller units (MCUs) include built-in hardware cryptographic accelerators that make encryption easy and fast.

The recent rise in government regulations on and certification programs for IoT security [4] [5] increasingly block access to important markets unless products are security certified.

Customers, who have consistently said they want more secure IoT products, are learning to look for security certification labels when shopping and say that they are willing to pay more for products that disclose strong security practices using those labels [6].

Manufacturers are learning that patching a security problem when products are in the field is much more costly than fixing that problem when the products is in development or design [7].

New IoT security standards, like Matter [8], take a fresh look at IoT security. Because of this fresh approach, Matter includes features that support all of the ZTA principles.

With these trends, embedded systems developers are finding that strong security is no longer an expensive and thankless task but rather a best practice that boosts sales with little extra expense.

ZERO TRUST ARCHITECTURE SUPPORT IN MATTER

The Matter standard illustrates how the ZTA principles are implemented in modern IoT products:

1. No network is trusted so all communications are secured.
   The Matter standard defines secured communication protocols for typical IoT transactions and enable extensibility to support new transactions.
2. All parties (users, services, devices, etc.) are authenticated.
   Every Matter participant has a unique identity that is cryptographically authenticated in the Matter protocols.
3. Dynamic policies determine which access requests are authorized.
   Matter defines a standard format for access control policies and a standard method for dynamically revising those policies.
4. The security of users, networks, devices, software, and data is monitored.
   Matter includes hooks that can be used to add security monitoring capabilities, but these are not enabled by default. IoT owners must opt in.
5. When a problem is detected, it is addressed promptly.
   As noted, Matter enables administrators to immediately revoke access privileges.

For embedded systems, Matter embodies the principles of ZTA. Now consider how it can be implemented in IoT and in embedded systems more broadly.

DESIGNING ZERO TRUST INTO IOT AND EMBEDDED SYSTEMS

Creating secured IoT and embedded systems requires a modernized, forward-looking approach. Consider each ZTA principle:

1. No network is trusted so all communications are secured.
   Using the Matter standard is a good way to get secured communication protocols. However, this is not always practical. In that case, choose a well-vetted, secured communication protocol such as DTLS or TLS.
2. All parties (users, services, devices, etc.) are authenticated.
   MCUs can support cryptographic authentication with a unique identity. For example, Infineon's OPTIGA Trust M secure elements support this approach, including optional support for Matter. Infineon's PSOC MCUs include Cloud ID which provides a unique identity and supports Matter connectivity.
3. Dynamic policies determine which access requests are authorized.
   Dynamic policies are implemented in application layer software. The open source Matter SDK (which runs on the PSOC 62S2 Wi-Fi BT Pioneer Kit and other platforms) includes support for these policies.
4. The security of users, networks, devices, software, and data is monitored.
   Security monitoring can be performed in the cloud, on a network gateway, or in devices. When performing system design, choose the optimal implementation strategy for each design.
5. When a problem is detected, it is addressed promptly.
   To enable rapid response to security problems, build in remote management capabilities to revoke authorizations if a hostile user, device, or service is detected. Then move on to recovery by including cyber resilience capabilities such as a recovery mode that automatically downloads and installs the latest firmware.

THE ROLE OF AI IN ZERO TRUST ARCHITECTURES

With the role of artificial intelligence (AI) increasing in both good guy and bad guy efforts, AI impacts ZTA in a few ways. One technique that has been used for more than a decade through neural networks (NNs) is constant monitoring. Instead of human monitoring, AI can thoroughly analyze logs, especially as a first pass. Training the NN to identify a suspicious vs. a not suspicious connection initiates the process. If the NN identifies a questionable situation, it can flag the user or person responsible for dealing with unauthorized access. This is most often associated with unauthorized credit card usage. Network intrusion detection also uses NN/AI for full-time monitoring for suspicious traffic. AI use in these types of applications will only increase and improve with increasing AI capabilities.

AI can initiate (be an active participant) by sending messages. In these instances, the AI itself needs authentication to ensure that it can be trusted just like any other user. For example, in a smart home, an AI node may automatically turn on lights as the sun sets or initiate some other welcome home action. As with other ZT aspects, the AI with its unusual capabilities needs to have fine-grained access control to avoid worst case scenarios. The rules must be established so AI knows its place and stays within its lane in the system.

IMPLEMENTING ZERO TRUST IN LEGACY IOT

Many enterprises still operate in the old security structure/model and transitioning to Zero Trust is not as simple as flipping a switch. As with any aspect of cybersecurity, implementation is all about risk. The highest risk situations need to be protected first. Higher risk items could be the payroll system, finance system, and/or intellectual property for an IT system or manufacturing company. No matter what the priorities are, they need to be identified and ranked to provide the right focus for the high-risk assets.

One of the first areas to address could be system administrators that typically have a very high level of access. In many organizations, all the system administrators share the same password—the root password. For Zero Trust, this is very dangerous because it is impossible to know who specifically performed a particular action, after the action has been performed. A starting point for establishing Zero Trust would be providing system administrators with their own unique credentials. Then, multi-factor authentication should be implemented to protect the administrator from cyber attackers. This type of attack is how the Ukrainian power grid was hacked in 2016—by obtaining the access password from a system administrator's computer through a key-stroke logger.

Further protection for high-risk assets can be accomplished by always requiring mutual authentication for high-risk systems and having fine-grained access control—access to only the systems and operations that each person needs to do their job. No one is inherently trusted, so access must be carefully controlled and restricted.

ABOUT THE AUTHOR

Steve Hanna is a Distinguished Engineer at Infineon Technologies. On a global basis, he is responsible for IoT security strategy and technology. As a security expert, Steve is a leader in many standards groups. Within the Connectivity Standards Alliance, he is a Director on the Alliance's Board, a member of the Executive Committee, the Chair of the Product Security Working Group, and a leader of several Matter Working Group security teams. Steve is also an inventor or co-inventor on 48 issued patents, the author of innumerable standards and white papers, and a regular speaker at industry events. He holds a Bachelor's degree in Computer Science from Harvard University.

Additional materials from the author are available at: www.circuitcellar.com/article-materials
References [1] to [10] as marked in the article can be found there.

RESOURCES

Infineon | www.infineon.com

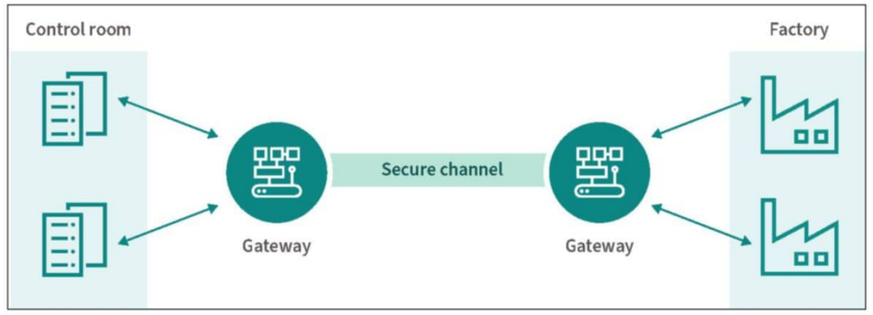


FIGURE 1
Gateways provide conduits from an untrusted zone to a trusted, protected zone for secure access to legacy systems.

Some legacy systems or subsystems cannot support ZTA and cannot be upgraded to do so. These systems should be put into a protected zone so that the only way to access them is through a firewall or security gateway. That gateway can implement all of the ZTA principles and protect the legacy system.

In some instances, two legacy devices in two different locations may need to connect to each other. For example, this could occur in industrial systems, where the control room must communicate across a distance with some legacy manufacturing equipment and neither supports a modern security protocol. In this case, they can be connected by a secure communications channel, with a gateway in front of each legacy device as shown in **Figure 1**. In this manner, the control room can communicate securely with the legacy manufacturing equipment even though both legacy systems are several decades old.

TRUST INFINEON FOR IOT SECURITY AND ZTA

All of the ZTA aspects can be implemented with Infineon product lines such as the OPTIGA Trust M, OPTIGA Authenticate, and PSOC 64 line/64 Secure MCUs. As shown in **Figure 2**, the OPTIGA Trust M can easily and securely interface with a PSOC 6 device, such as the KBA23540. This applies to any PSOC 6 device and uses ModusToolbox software. Evaluation kits are available to support this process.

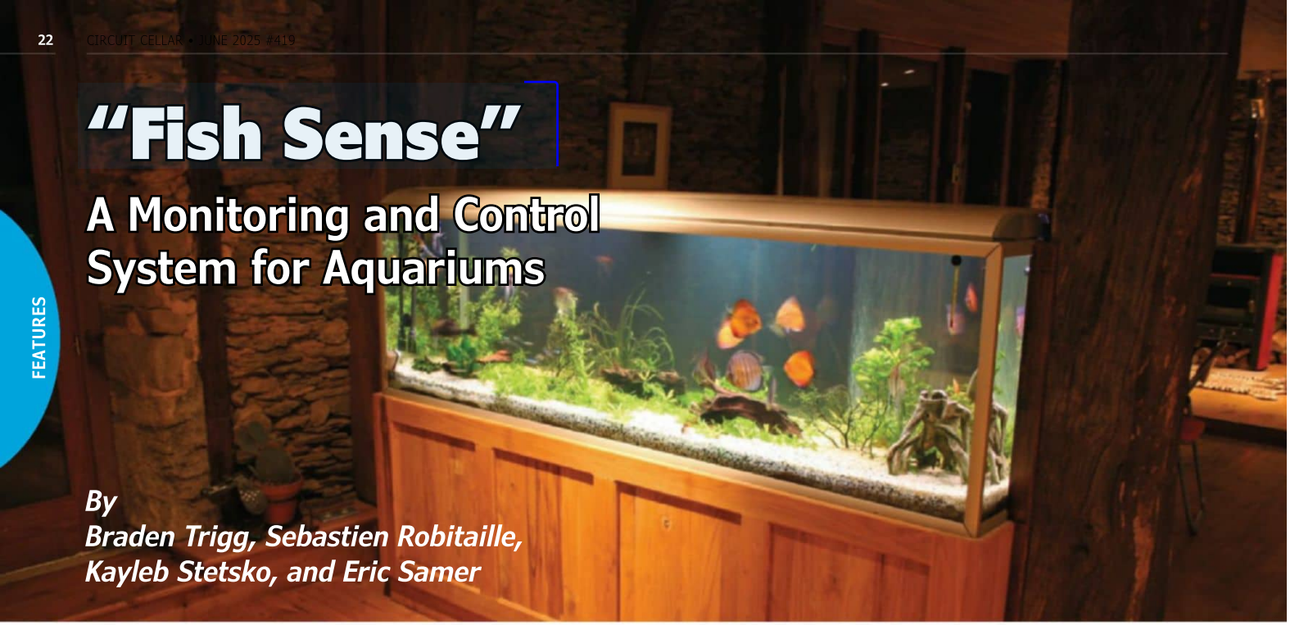
OPTIGA Trust M is the first security platform to receive the Cyber Security Agency of Singapore CLS-Ready certification [10]. Another embedded security option is the OPTIGA Trust M MTR—a special version of the OPTIGA Trust M that includes support for the Matter protocol.

Finally, the new OPTIGA Trust M IoT security kit provides the easiest way to develop and evaluate end-to-end security use cases for IoT devices. The kit enables customers to prototype various security use cases and build full-featured IoT applications.

MAKING THE TRANSITION TO A NEW TRUSTED ENVIRONMENT

Zero Trust Architecture is a groundbreaking approach to establish trust for users, devices, and software. With Zero Trust, security must be taken more seriously than ever before. In addition to industry associations providing direction and capabilities to establish greater trust, such as CSA's Matter standard, when governments enter the picture, the need to implement their requirements rises to the next level. Fortunately, there are hardware-based solutions available now. For embedded systems designers who are ready to implement improved security, controllers and development tools in a variety of trusted families already exist and are continuously expanding to improve trust in IoT connectivity.

FIGURE 2
Connecting the host microcontroller PSOC 6 to Shield2Go Security OPTIGA Trust M via a shielded I^2C interface provides an additional layer of security [9].



"Fish Sense"

A Monitoring and Control System for Aquariums

By
Braden Trigg, Sebastien Robitaille, Kayleb Stetsko, and Eric Samer

Keeping fishes and invertebrates in captivity is always a challenge because so many critical environmental factors need to be carefully controlled. Using ESP32s, CAN Bus, MQTT, and Ethernet, these four Camosun College students developed "Fish Sense," a modular, expandable method of monitoring conditions in a home aquarium, automatically notifying the aquarium keeper when something goes awry, and even correcting some problems.

Keeping aquatic life alive in an aquarium is an exercise in frustration tolerance—there's always something going wrong. The challenge is finding the problem in time, and balancing the ecosystem before things begin cascading in the wrong direction. Otherwise, before you know it, you're stuck with whatever living creatures you've managed to save, living out of sight in the aquarium's sump tank, while you attempt to figure out what to do with 85 gallons of dirty saltwater (**Figure 1**)—not to mention the hours draining and refilling the tank can take.

Kayleb has been keeping fish and corals for most of his life, and is aware of the many struggles associated with aquariums, having had to reset his tank multiple times. Saltwater aquariums are far more complex than freshwater systems, especially for growing corals. They can require a secondary "sump" tank (**Figure 2**) to act as a refugium—somewhere that delicate algae and other organisms that serve as food can reproduce. As filter feeders, corals that are housed in the display tank need a constant current in the water to enable them to feed on the organisms from the sump tank. Also, to truly thrive, corals require lighting and a narrow range of water quality parameters—including temperature, salinity, dissolved oxygen, and pH.

Disaster is an opportunist, and water being pumped between tanks provides ample opportunity for it to strike. Although leak- and water-level sensing can alert you to an issue; keeping track of everything and controlling the pumps manually can be both challenging and time consuming. Being tech savvy, Kayleb has automated some of these tasks. But with only hobby-level knowledge, he had previously been able to implement only hobby-level solutions.

When the four of us began discussing ideas for our capstone project for our electronics technologist diploma at Camosun College, creating an aquarium monitoring and control system seemed like a natural choice. We were given a full semester for this project, starting in September 2024. But wanting it to be more than "good enough," we got to work at the start of the year.

After researching the competition, we assembled a list of concerns with the existing aquarium-monitoring solutions on the market. Almost all were a single unit, rigid in what they did, and didn't allow for easy expandability or partial systems; you really had to commit financially (usually to the tune of thousands of dollars).

So, to set our product apart from the rest, we focused on "modularity" and "expandability."

Modularity allows for customization. Each ecosystem will have its own requirements, and each fish or invertebrate keeper has their own list of priorities. We needed our system to be user defined in terms of what devices and components keepers could add—not just a black box of "this is what it does," but several black boxes that can be used together.

Expandability allows for growth. Having gone through several tank expansions himself, Kayleb was all too familiar with how a simple fish tank can grow into something much bigger. (Remember that 85-gallon tank—that was after his wife insisted on cutting back!) We built our system to allow users to enlarge their tank without being held back by the technology.

With these two tenets to guide us forward, we decided to just build a system to monitor and control everything, simple to use, expandable, and completely modular. Easy, right?

Our first real task was making the decisions about what technology would make our project, which we named "Fish Sense," a reality: ESP32s, CAN Bus, MQTT, Ethernet cords, and more. Many summer evenings were spent digging through and comparing datasheets.

Code for the various Fish Sense components is available on the GitHub ProjectFishWorks web pages [1].

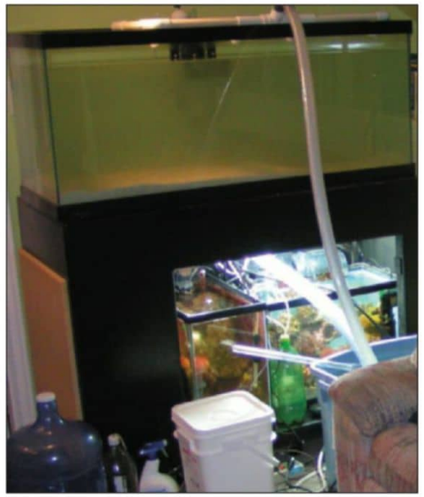


FIGURE 1
Kayleb's saltwater fish tank after a collapse. He is just getting ready to drain it. Everything salvageable is below in the sump tank, but it's a grim and heartbreaking scene. Without constant monitoring, it can be a gamble knowing what's happening inside.

SYSTEM DESIGN

To test Fish sense, we put together a basic saltwater aquarium that we called the demo tank (**Figure 3**). This setup consists of a 20-gallon main display tank, with a 25-gallon sump tank below. The sump tank houses a population of algae that allows us to filter the water that is being circulated between the display and the sump tanks.

In addition to the circulating pump, we used a device called a wave-maker in the display tank to create some additional water movement. This is especially important for corals, as they use this water current to feed. The saltwater in both tanks was created by mixing reverse osmosis water with a specific amount of sea-salt mixture to recreate our own take on natural saltwater.

To satisfy the modularity and expandability requirements for the project, first, we decided that each device in the system would have its own microcontroller. (The devices we used are discussed later in this article.) This would allow each device to operate independently, giving users the ability to easily pick and choose what devices they want in their system. Second, we decided that a wired network would be preferable, since we didn't want to deal with connecting so many devices to Wi-Fi or having to power them individually. Third, we wanted the system to be accessible from a mobile app or something similar, allowing users to monitor and control their system from anywhere.

With this system topology (**Figure 4**) decided upon, we started fleshing out the technologies that the system would comprise. We decided on two custom circuit boards for the system.

Microprocessors: Our node controller would act as the general-purpose controller for each device. It needed enough GPIO to connect to various peripherals (sensors,

FIGURE 2
A sump tank is a refugium, somewhere for nutrient providing organixms to thrive and feed the life in the display tank. This is Kayleb's current sump tank, which also houses the various pumps and other equipment that keep the tank healthy.

controllers, and so on). It also needed some way to connect to our second board, the base station. The base station would act as the controller for the system; it would have a Wi-Fi connection, and would interface all the devices with the mobile app.

We settled on the ESP32 family of microprocessors for these two boards. The ESP32s are available in module form, with all the critical circuitry—such as flash and the RF circuits—packaged together, which simplified our PCB layouts.

FIGURE 3
We are extremely proud of our completed demonstration tank. Almost every aspect of it is controlled by Fish Sense, with even more controls planned for the future.

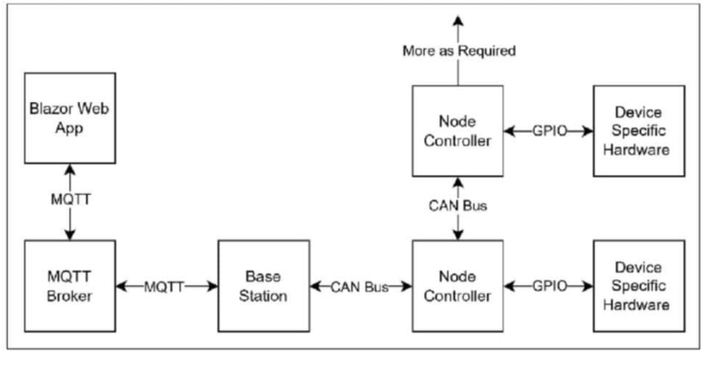


FIGURE 4
High-level block diagram of Fish Sense. Device-specific hardware interfaces directly with our node controllers, which use CAN Bus to communicate with the rest of the network. Our base station, at the heart of it all, uses MQTT to reach out to our web app.

CAN Bus: Next, we needed some way to connect the devices together and to the base station. I²C was briefly considered, but abandoned, due to its range restrictions and limiting controller target topology. After further research, we landed on a controller area network (CAN) Bus, developed by Bosh in the 1990s, which is a serial communication protocol that includes built-in addressing and collision detection/avoidance. CAN Bus also has a longer range than I²C (hundreds to thousands of meters). While this was a longer range than we needed in the short term, having the flexibility of creating larger and more distributed systems would leave our options open for future versions of this product.

Cables: Once we had a protocol picked out, we needed a physical layer to run the CAN Bus signal over. We wanted something reasonably durable and easy for the user to plug and unplug. Standard network cables, consisting of CAT 5 or 6 cable with RJ45 terminations, emerged as a clear choice for us (**Figure 5**). These cables are super easy to use and had a close enough characteristic impedance to the CAN Bus standard. They also have plenty of conductor pairs, one of which we used for the CAN Bus signal and the rest to supply power between connected devices.

Mobile App: The last piece of the puzzle was the mobile app. To limit our development complexity, we decided to create a web-based app that would run directly in a browser. Running in a browser would allow us to target both mobile and desktop environments from a single codebase. We settled on the Blazor framework from Microsoft. Blazor is a part of ASP.NET, and allows for the creation of interactive web UIs using C# alongside HTML and CSS. We all had experience programming in C# from our studies at college, so we felt that it was a good fit for our team.

MQTT Protocol: To facilitate the transfer of data between the base station and our web app, we chose the Message Queuing Telemetry Transport (MQTT) protocol. MQTT is a publish-subscribe network protocol usually used in bandwidth-limited situations, such IoT devices. The MQTT protocol involves an intermediate server called a broker. Clients—in our case, the base station and the web app—send messages to the broker on topics and can subscribe to topics to receive messages from other clients. We were able to use a free broker from HiveMQ.com. However, HiveMQ only has a 3-day message retention period, so we are exploring different solutions to allow for long-term use.

This brought us to September, when we started working on the project full time. With the major details decided on, we turned to designing our two custom circuit boards.

NODE CONTROLLER

The node controller (**Figure 6**) is a fairly simple board. The ESP32 C3 that we chose for the board has a dedicated CAN Bus peripheral (called the TWAI driver), but it still requires an external transceiver chip to create the differential CAN Bus signal. We used the Microchip MCP2551 IC for this purpose. (The MCP2551 is now discontinued, but similar devices, such as the MCP2561, exist). For GPIO, we simply broke out the 13 remaining pins not used for the CAN Bus transceiver to header pins. These pins allowed us to create protoboard "hats" to add any device-specific circuits to the node.

For power, we ended up with a three-rail setup. We receive 24V DC from the base station via the RJ45 plug. This voltage is then reduced to 5V using an integrated switching regulator to power the CAN Bus transceiver. Next, a linear regulator drops that 5V to the 3.3V required to power the ESP32 and most peripherals. (Some sensors and other devices use 5V, though, so it's nice to have as well.) We used the ESP32-C3-WROOM-02 module with an included PCB antenna. Although we didn't end up using Wi-Fi or Bluetooth in our final version of the project, the cost was the same as the module without the antenna, and we wanted to leave our options open for the future.

Firmware on the node controllers is also not overly complex. Sebastien, our programming wizard, decided that using C++ and the Arduino framework would enable us to use the existing Arduino libraries for various hardware interfacing, saving us a lot of firmware development time. We created a custom library to handle all the shared CAN Bus functionality. This library contained a function to send data to the CAN Bus network and a callback function that would be called every time the base station or another device on the network transmits data. Using this library, the node controller can send data to the base station, which then sends it to the web app. The node can also receive data, either from another device, or from the web app via the base station. Code for the node controller and other devices is available on the GitHub ProjectFishWorks web pages [1]

BASE STATION

The base station (**Figure 7**) is the main interface for our devices to connect to the web app. With the base station acting as a hub for the system, we also decided to add some extra functionality to it. First, we wanted to notify the user if there was an issue with the system (see message in the display, Figure 7) if, for example, a leak was detected, or the water temperature was too high. We added RGB LEDs and a buzzer to alert users nearby, and also gave the base station the ability to send emails to users if they were away from home. We also added two buttons and a small OLED screen to provide a basic user interface for configuring options, such as the Wi-Fi connection. The final function that we tasked the base station with was to log data from the devices to an SD card, so historical data could be accessed from the web app.

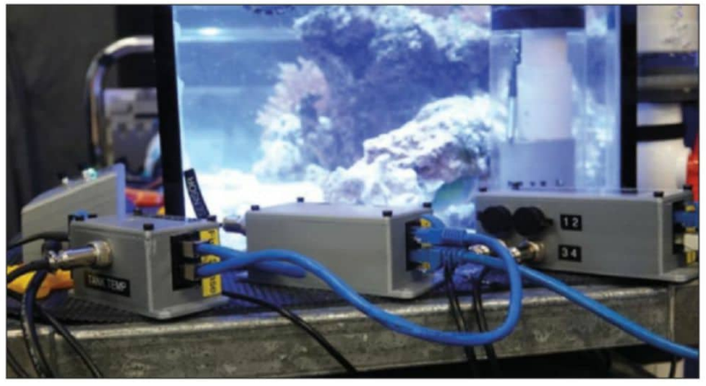


FIGURE 5
Regular Ethernet cables proved to be the best choice for carrying our data signal and power. They have enough conductors to carry the CAN Bus signal and power throughout the device network.

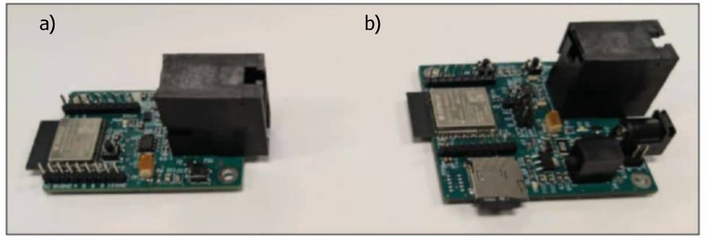


FIGURE 6
Our two custom PCBs. (a) Node controller with an ESP32 C3. (b) The more powerful base station, with the ESP32 S3, SD card slot, and barrel jack to connect to power.

FIGURE 7
Loud and bright, we ensured that our base station would grab the user's attention if there was something wrong with the tank. It can also send an email alert if the users are away from home.

FIGURE 8
Here are the devices we chose for the first version of Fish Sense. Starting from the bottom right, going clockwise: pH Sensing, the Base Station, Leak Detection, AC Control, Lighting, and Temperature Sensing.

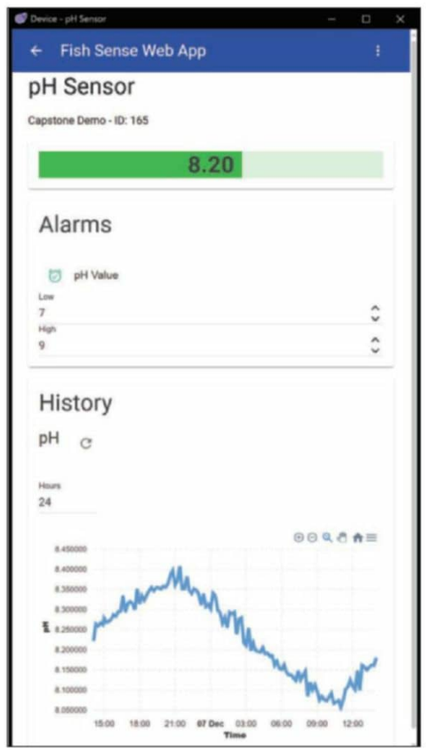


FIGURE 9
The Fish Sense Web App, showing the details page of our pH Sensor. We have built-in user controllable alarms for highs and lows, and up to three days of history with a scalable graph view.

As one would imagine, these extra features required a more complex PCB design, and a more powerful processor. We chose the larger and more powerful ESP32-S3-WROOM-2 module. This module gave us more GPIO to work with, as well as a dual-core processor that would allow us to split the slower Wi-Fi and SD card operations from the more real-time CAN Bus operations.

The power supply for the base station is mostly the same as the one for the node controller, except that the 24V power is provided by an external AC-DC adapter and barrel jack. We also put the 24V power through a high side power switch (ITS428L2), to give us better control of the network boot-up timing and a current/power monitoring IC (INA219) to monitor device network power usage.

Having not finished our user interface decisions when designing the board, we opted to just break out all the GPIO, with some paired with their own ground and 5V pins to power the LEDs or other potential peripherals.

Finally, during some late nights and with the help of our instructors, we finished the board designs using Altium Designer, and had them manufactured by JLC PCB. We assembled the boards with a solder-paste stencil, our school's manual pick-and-place machine, and a vapor phase reflow oven. While there were a few minor issues with our boards (and a few bodges required), we made it through without needing any revisions. There are many things we would like to improve with these boards, with plans already in the works for subsequent revisions.

Data Flow Between CAN Bus and Web App: With hardware completed, we needed to map out how the base station's firmware would handle data flow between the CAN Bus network and our web app. The easiest way to break it down is to follow the example of the path for a temperature reading.

Starting in the device's node controller firmware, we receive a floating-point number from the temperature sensor. CAN Bus messages are broken up into two main fields, giving us 8 bytes for our payload and 29 bits for an identifier. We use the identifier field to transmit the node's unique 8-bit ID, as well as a 16-bit message ID that indicates what data is being sent. The temperature value goes in the payload field, and this message is then transmitted over the CAN Bus network to be received by the base station.

Once the base station receives the message, an MQTT topic is created with the node and message IDs and a unique ID for the base station. The temperature data from the CAN Bus message, along with the time the message was received, is added to the MQTT message payload and sent to the broker. The web app can now subscribe to this topic on the broker and receive up-to-date temperature readings.

This whole process is repeated in reverse when we want to move data from the app to the devices, such as when we want to modify a temperature alarm setpoint.

DEVICES

By October, we finally had what you could call the "brains" of the system in the base station, and a network of nodes to act as a "nervous system;" therefore, it was time to start incorporating the "extremities," our devices (**Figure 8**).

The first device that we started working on was our aquarium lighting control system. To ensure the comfort of our corals, we used two strings of high-intensity LEDs—one white and one blue. Both strings contain eight series LEDs driven by MEAN WELL constant current LED drivers. We dim the LED drivers using a 0 to 10V PWM signal from a node controller, allowing us to create a customizable day-to-night routine and manually set brightness in the web app.

While there is an almost endless number of parameters that could be monitored in an aquarium, we decided that leak detection, water level, water temperature and pH, along with air temperature and humidity, would be a good starting point. All these parameters can be monitored on the web app. Further, users can configure high and low alarm setpoints, so that an alarm can be set off if any of the parameters are out of range (**Figure 9**).

Smart Power Bar: Because an alarm going off doesn't do much to mitigate the potential danger, Eric, our team's Red Seal Electrician, designed and constructed an AC outlet control device. Our "smart power bar" would allow the system to take action in response to these alarms automatically toggling outlets as required. For example, it would turn off the pump if the water level was too high, or turn on a heater if the temperature was low.

We placed all the components for the device (outlets, control relays, and the node controller) in a divided five-gang electrical box. The divider meant that we could provide some physical isolation between the AC outlets and relays, and the lower-voltage node controller. We also chose a dual-channel relay module with a built-in AC to DC power supply for the relay coils. Using opto-isolators for the relay control signals meant we could

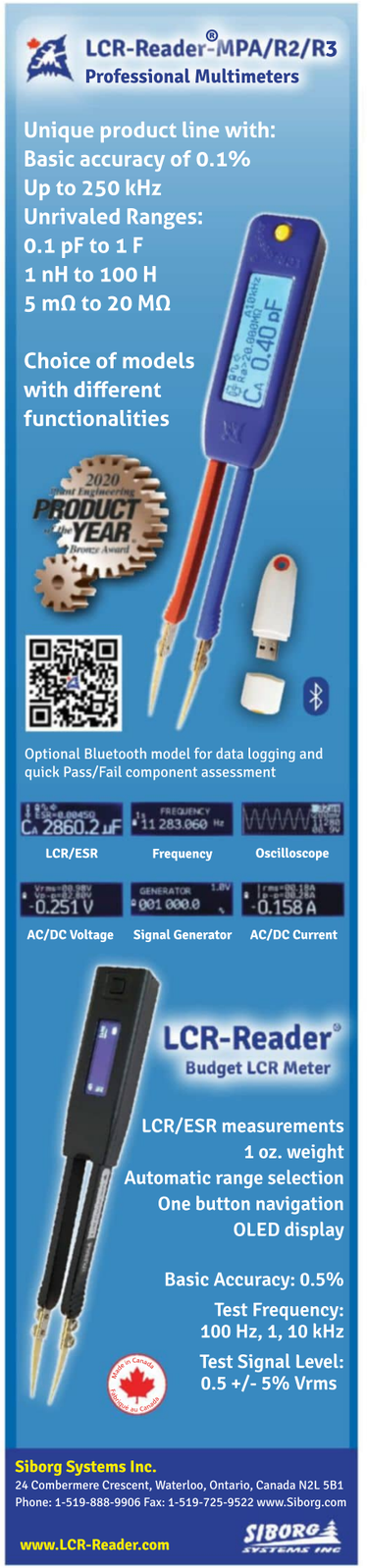
ABOUT THE AUTHORS

Kayleb Stetsko is the visionary with a lifelong love for fishkeeping. Above all he is a loving father, who would do anything for his children. Beyond that, he treats anything alive like his own, including the fish of the world. His love for technology and engineering got him to where he is today. His beard is legendary. Email: kayleb_s@hotmail.com

Eric Samer is a Red Seal Electrician focused on safe and effective power solutions. Eric worked as a Red Seal Electrician until an accident forced him off his feet and he has since decided to follow his love of technology. He has a huge passion for space and science (and his cats). His beard is also legendary. Email: eric.samer@gmail.com

Sebastien Robitaille is our programming expert, skilled in integrating complex systems. He is the youngest of the group, but probably the smartest and his drive to learn and get things done is probably going to change the world. In his spare time, he helps track the bird populations and plays with computers. His beard, by the way, is legendary. Email: sebastien@robitaille.info

Braden Trigg is the organizer and writer, translating technical details into clear communication. After spending 16 years in hospitality working as a Chef he has since moved into IT. Braden is a passionate video gamer and is looking forward to furthering his passion in electronics, computers, and science in general. He can't grow a beard but is still legendary. Email: triggbraden@yahoo.ca



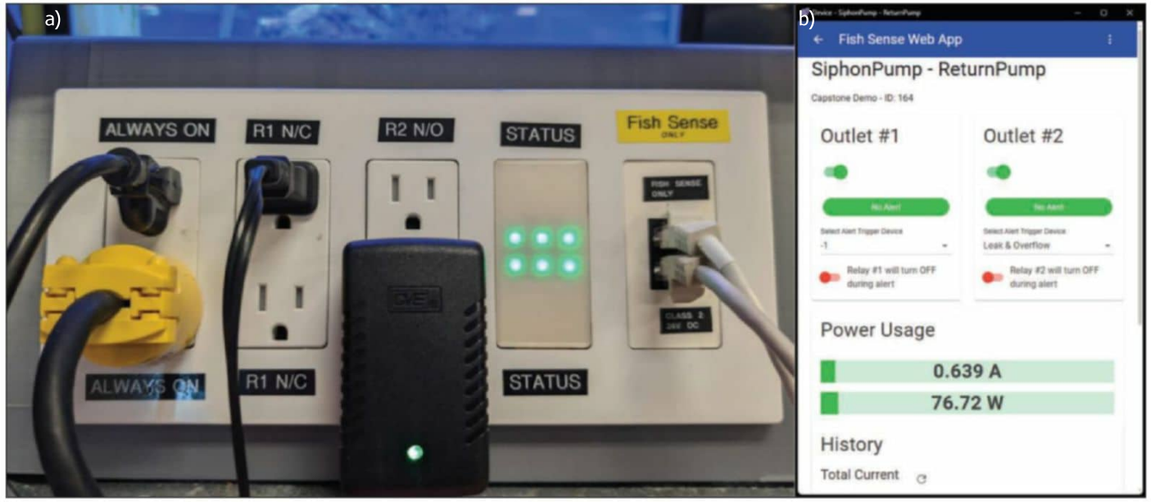


FIGURE 10
(a) The finished AC Control Device provides controllable outlets for our tank. (b) The web app device page allows users to configure how alarms affect each outlet and view the power consumption of the system.

fully electrically isolate the node controller from the relays and outlets. Finally, a GFCI cord end was used, since the device would be near water.

We designed and 3D printed a bracket to hold the node controller in one blank cover plate, and some RGB LEDs—to act as status indicators—in a second one. We chose a form factor that allowed everything to fit into any 3.5"-deep electrical box, leaving our options open for in-wall mounting in the future.

Eric constructed the AC outlet device with three pairs of outlets: one always on, one connected to a normally open relay, and one connected to a normally closed relay. This assortment of outlet choice allows the user to decide whether a given outlet is on or off in the event of a control system failure. We also added a current clamp, so that the power usage of all the connected AC devices could be monitored in the web app.

Because the AC outlet is arguably our most complex device, it is still in the early stages of development. Although we were able to complete the current version (**Figure 10**), in the future, we would like to add more outlet configurations, per-outlet power monitoring, and more complex alert triggering and scheduling options in the web app.

Code and other information about the various Fish Sense components is available on the GitHub ProjectFishWorks web pages [1].

Additional materials from the author are available at: www.circuitcellar.com/article-materials

References [1] as marked in the article can be found there.

CONCLUSION

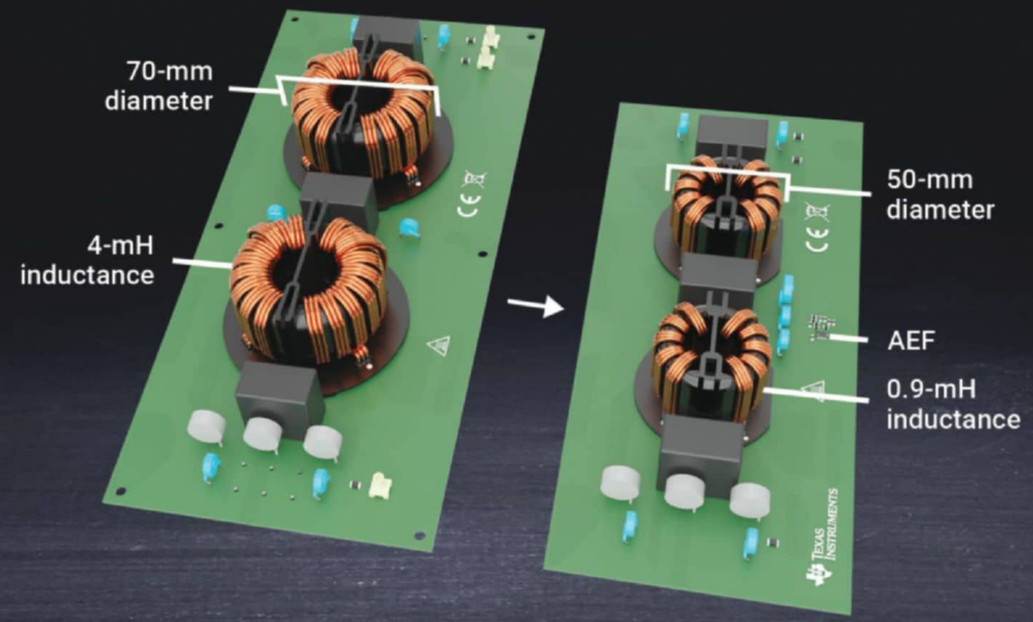
From the very beginning, Kayleb's dream of a completely modular solution for his aquariums promised to be not only an exciting and fun project, but also something that could truly help make the joy of aquarium keeping accessible to a larger audience. In 12 months, we put our all into Fish Sense, and we believe we made something truly novel. We set out to simplify the aquarium experience, making it as easy to grow your monitoring system as it is to grow your aquarium. With the ease of set up and the modular, expandable design we achieved, we believe we accomplished this goal.

That said, we would like to continue to improve our product, Fish Sense, and have many plans for the future, in addition to those mentioned earlier. We would like to implement more devices, such as the Atlas Scientific line of environmental sensors, a variable-speed pump controller, and an automatic feeder. Furthermore, we aim to make improvements to our existing technology, including a full redesign of the lighting system and new versions of the node controller and base station PCBs.

We also envision many other use cases for Fish Sense beyond home aquariums, as its versatility and modular design open up a wide range of possibilities. From hydroponics in agriculture, where precise environmental monitoring can enhance crop yields, to fish and shellfish farms that require reliable water quality management, our system is adaptable to meet diverse needs.

Until then, we know that the fishes and corals in our demo tank (Figure 3) will live the easy life in their tank managed by Fish Sense.

Meet stringent EMI requirements and cut choke size in half.



Our new portfolio of stand-alone active EMI filter (AEF) ICs help designers meet stringent EMI requirements while reducing system size, weight and cost for single- and three-phase AC/DC systems. The TPSF12C1/-Q1 and TPSF12C3/-Q1 allow engineers to shrink the value of common-mode chokes by up to 80% inductance, resulting in over 50% smaller size compared to purely passive filter solutions. Meet your EMI performance standards and increase power density with AEF ICs today.

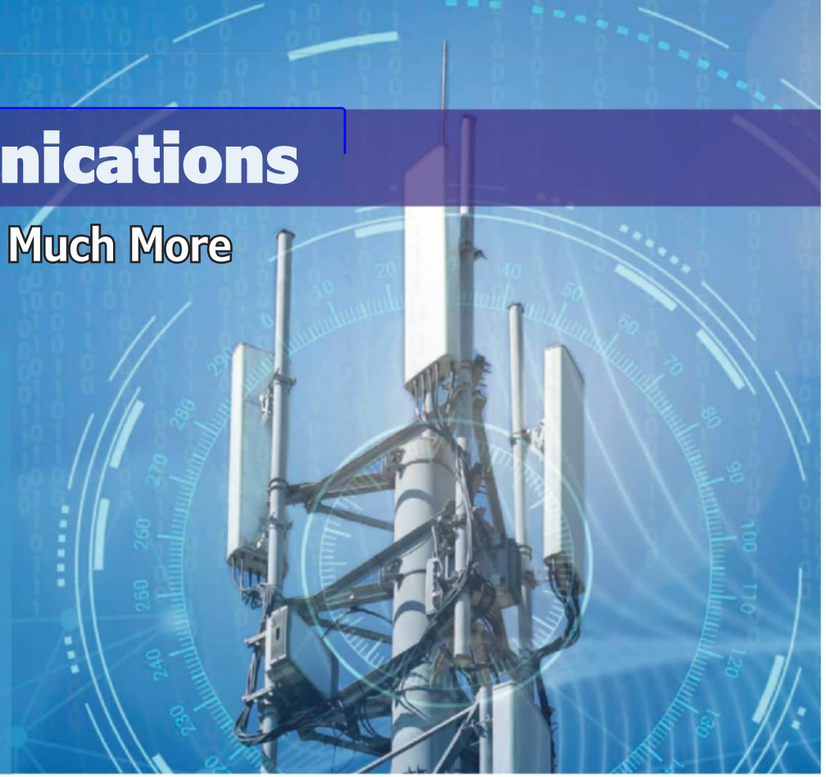
Learn more > www.TI.com/AEF

Texas Instruments

TECH FEATURE

Telecommunications

Bandwidth and So Much More



Telecommunications place high demands on network performance but complex applications mean processors, sensors, and system management components are often cutting-edge, as well. Understand which components are critical from edge device to network core to give telecom-enabled systems and applications a solid design boost.

By
Curtis Franklin

Telecommunications is a single industry that encompasses an exceptionally broad range of products and services. From carrier-grade core switches to "feature phones" (cell phones that would have been at home in 2002), telecommunications products encompass products in which processing speed and data throughput are paramount to those in which size and power consumption are the primary considerations.

Packaging and processing power together are driving a great deal of innovation among processor and network control components as device manufacturers seek to produce ever more powerful edge devices in ever smaller packages. These considerations, along with non-technical issues like international supply chains and the impact of outside events on the health of those brittle chains, have created a market in which new suppliers appear while incumbent suppliers add to their product lines and integration capabilities.

CHANGE DRIVERS

The telecommunications industry is undergoing rapid transformation, driven by customer demand and advancements in embedded electronics. The demands are consistent: More capabilities at reduced prices. The technical advancements are enhancing network efficiency, improving connectivity, and enabling new applications in mobile and broadband communication.

In addition to the customer-driven evolution already mentioned, there are six significant trends promising to drive innovation and change in telecommunications embedded electronics in the coming months.

Edge Computing and AI Integration

One of the most significant trends in embedded electronics across all market segments is the integration of edge computing and artificial intelligence (AI) and telecommunications is no exception. Traditional telecom networks rely on centralized data centers, but edge computing shifts processing closer to the source, reducing latency and improving performance in distributed applications. AI-powered edge devices can optimize network traffic, enhance security, and enable predictive maintenance, making telecom infrastructure more efficient.

Advanced Cybersecurity Measures

With the rise of Internet of Things (IoT) devices within telecommunications, security has become a top priority. Embedded systems now incorporate multiple layers of cybersecurity, including secure boot processes, authentication protocols, zero-trust architectures, and hardware encryption. These measures help protect sensitive data and prevent cyber threats, ensuring reliable and secure communication networks. As these advanced cybersecurity measures become accepted as part of telecom-enabled IoT edge devices the functions will increasingly be integrated directly into hardware rather than being implemented solely through software.

5G, 6G, and Beyond

5G network deployment and adoption has revolutionized telecommunications, enabling faster data speeds and lower latency. Embedded electronics play a crucial role in supporting 5G infrastructure, with specialized RF components, network interface chips, and optical modules designed for high-speed connectivity. Research into 6G technology is already underway, promising even greater advancements in wireless communication. And with both cellular and non-cellular telecommunications, software-defined networking is adding capabilities while requiring more from processors and support circuitry.

AI-Driven Network Optimization

Telecom providers are leveraging agentic AI (artificial intelligence that uses large language model AI as a piece of its technology but focuses on decision making rather than content creation) to optimize network performance. AI-driven systems can autonomously manage network resources, adjust bandwidth allocation, and enhance signal quality. While still in the early stages, agentic AI is showing the potential to reduce operational costs and improve user experience while making telecom networks more adaptive and resilient.

Smart Connectivity and IoT Expansion

The expansion of telecom-enabled IoT ecosystems is driving demand for embedded electronics that support seamless connectivity. Telecommunications devices now feature low-power sensors, advanced interface chips, and efficient power management components to enable smart applications. These innovations are especially important for industries such as healthcare, transportation, and smart cities that depend on massive sensor data to be shared with central back-end processors to make control decisions in real time and enable better policies with human analysis.

Optical and Quantum Communication

Advancements in optical modules are enabling increasingly high-speed data transmission over fiber networks. These advancements include both drivers that optimize available bandwidth in single-mode long-distance fiberoptic cable and transmitting units that reduce the need for optical amplifiers at regular intervals in transoceanic an other long-haul networks. With private companies showing a willingness to commission low-latency networks with the goal of shaving milliseconds from transactions, amplifier minimization has significant promise for all coming network deployments.

In addition to advances in fiberoptics, research into quantum communication is gaining traction, offering ultra-secure data transfer methods. These technologies are expected to play a vital role in the future of telecommunications, though few today will speculate with any certainty on precisely what form that role will be.

THE CRITICAL COMPONENT COLLECTION

Telecommunications systems, from network cores to edge devices, rely on a broad ecosystem of embedded electronic components that enable signal processing data transmission, and connectivity. The major components within that ecosystem are:

Microcontrollers and Processors serve as the brain of telecom devices, conducting signal processing, system control, data routing, and application code execution..

Field-Programmable Gate Arrays (FPGA) are used in network switches and base stations for flexible and high-speed processing. FPGAs are frequently low-cost options for providing specific functions while adding security for intellectual property since code is locked into the silicon.

Application-Specific Integrated Circuits (ASIC) are designed for specialized tasks including encoding/decoding signals, application-specific protocols, and network routing.

Memory Modules include RAM for temporary data storage and Flash memory for firmware and system configurations.

RF Components such as antennas, amplifiers, and transceivers handle wireless communication. Some components (e.g., antennas) may drive product packaging because of their placement requirements in the finished units.

Optical Modules, including fiber optic transceivers and wavelength division multiplexers, enable high-speed data transmission over fiber networks. Optical modules can have a significant impact on the overall performance and transmission reliability of a switch or other solution and must be carefully matched to the system requirements.

Power Management ICs regulate and distribute power efficiently within edge devices and telecom infrastructure components. Power management is a radically different proposition at the two ends of the telecom

network architecture: At the core, power management can save an operator millions of operating dollars and dramatically increase the reliability of the network. At the edge, customers demand longer run times between battery recharges and are sensitive to factors such as heat management and battery longevity within the device.

Sensors and Interfaces handle tasks such as environmental monitoring, signal integrity, health monitoring, and user interactions in edge devices. A sensor package may well be the entire reason for a telecom-enabled IoT device's existence. And sensors are critical as part of the system monitoring and management package in network switches from the core to the campus.

Network Interface Chips support Ethernet, Wi-Fi, multi-protocol label switching (MPLS), and cellular network connectivity for data transfer within and between various networks. While some network interface chips are concerned solely with pushing packets (or bits) as quickly as possible, a growing number are built around sufficient processing power to enable software-defined networking, on-chip cybersecurity, and complex system health monitoring and management.

So which companies have products within each of these categories? In this article we focused on companies designers are most likely to encounter when working in North America. This is not a comprehensive global list but it does contain the companies you'll see when specifying components for projects from a bench in the US, Canada, or Mexico.

FIGURE 1
ARM Cortex processors use the same ARM architecture licensed for use in many processor designs. (Image courtesy arm limited)

PROCESSORS FOR TELECOMMUNICATIONS

Telecommunication processors are the most visible and well-known component of most telecommunications-based systems. While there are a number of different architectures, most processors are manufactured by one of a small handful of companies. Six manufacturers dominate the market.

Advanced Micro Devices (AMD) — Produces high-performance processors for data centers and telecom applications. AMD's processors, in both the EPYC and Ryzen families, are architecturally compatible with processors manufactured by Intel. Like Intel, AMD is now advertising its processors' suitability for applications that bring AI to embedded systems and network control. In these applications, the Ryzen is promoted as an energy-efficient option in which a small form factor and lower power consumption are critical considerations, while the EPYC is available for applications in which power consumption and small packaging are lesser considerations. The combination gives AMD options for both core network switching applications and edge devices in small consumer format or IoT deployments.

Intel — A dominant player in the semiconductor industry, Intel produces high-performance processors used in telecom infrastructure and networking. The most commonly used Intel processor for telecom applications is the Xeon 6 with up to 288 Efficient Cores (E-cores). With this processor, Intel claims 2.7 times the performance of earlier Xeon processors when used in a 5G rack. In addition, Intel's Infrastructure Power Manager (IPM) software promises significant power consumption reductions for servers using the Xeon processor.

Xeon processors are also available with up to 128 Performance Cores (P-cores) for applications in which compute power is the determining performance marker. P-core offerings are differentiated with hyperthreaded cores featuring built-in matrix engines that accelerate compute-intensive AI, HPC, and data services workloads, whereas E-core offerings omit matrix engines in favor of higher, single-threaded core density [1].

Microchip Technology — Focuses on processors that are, in the company's words, "Simple, Secure and Robust." The processors, based on the ARM architecture (**Figure 1**), support a variety of wireless protocols, from Bluetooth Low Energy to Zigbee. Microchip Technology emphasizes its products' ability to be included in rapid prototyping exercises, with designs that move quickly from concept to production [2].

NXP Semiconductors — Develops processors and microcontrollers for industrial, automotive, networking, and telecom solutions. NXP's Layerscape processors, part of its EdgeVerse edge-computing platform, offers a large matrix of features, capabilities, and connectivity options. Layerscape series processors, built on ARM core technology (Figure 1), extend different levels of performance to very small form factors. The various levels of functionality allow applications that stretch from power-constrained networking and industrial applications to software-defined networks and embedded systems requiring an advanced data path and varieties of peripheral communication interfaces [3].

Qualcomm — Specializes in mobile and wireless communication processors, including 5G-enabled chips, Qualcomm supplies chips, processors, and systems on a chip for a wide variety of telecommunications functions. Qualcomm's current processor family for telecommunications edge devices is the Snapdragon X series (**Figure 2**), with members of the series including the Snapdragon X, Snapdragon X Plus, Snapdragon X Elite, and Snapdragon X Plus 8-Core.

The various Snapdragon models vary in the number of cores (8 to 10) and accessory functions available on-chip, with networking, security, and AI processing among the functions available in different models [4].

Texas Instruments (TI) — Offers a range of microcontrollers and processors for embedded telecom systems (**Figure 3**). TI's offerings in telecommunications processors include products such as the MSP430F552x and the MSP430F551x mixed-signal microcontrollers. Each of these contains a variety of on-board connectivity features that suit them for deployment in IoT applications in remote locations; the processors are both powerful and come fitted with numerous connectivity options. In addition to these, TI is known for its significant presence in the DSP market, with DSPs found in a wide variety of applications that require fewer functions than might be supported by one of the MSP processors, but still has the need for high-speed, consistent performance. Texas Instruments is designer friendly with many different reference designs available to help kick-start a design process [5].

BEYOND THE PROCESSOR

Telecommunications solutions do not live and die by the processor alone. They require a constellation of support functions orbiting the processor, each function providing capabilities required by the application. Some of the components in the support constellation are specific to telecommunications, some are more broadly applicable functions that have significant use within telecommunications, and some split the difference between industry specific and general purpose.

CUSTOM CHIPS IN TELECOMMUNICATIONS

Custom silicon, including FPGAs and ASICs are heavily used in the telecommunications industry to provide product-specific functionality in cost- and space-effective packages. Among the different options for these custom chips, the system for developing and deploying the required custom programming is a genuine point of separation. Engineers and developers should look closely at each option and determine which development solution most closely meshes with their own workflow.

Five companies have significant roles in the market manufacturing FPGAs for telecommunications applications.

AMD — A market leader in FPGA technology, AMD offers high-performance solutions for telecom infrastructure, including 5G networks and data centers [6].

Efinix — Specializes in innovative FPGA solutions with a focus on efficiency and performance. Three different FPGA series — Titanium, Topaz, and Trion — offer engineers multiple options for the type of FPGA they want to build into their design, considering the number of gates, additional features on chip, and power consumption [7].

Infineon Technologies AG — A global semiconductor company that develops FPGAs for various industries, including telecommunications [8].

Intel — Provides FPGAs optimized for networking, cloud computing, and telecom applications. One of the points Intel makes about its FPGA offerings is product longevity: The company says that its Cyclone FPGA products will be available until 2040, with other FPGAs in the current product catalog available until 2035 [9].

FPGAs are not the only programmable devices found in telecom applications; ASICs are common components, as well. In general, if a telecom application requires field updates and adaptability, FPGAs are the go-to choice. But if high efficiency and optimized performance are more important, with field-reprogrammability a non-issue, then ASICs are preferred.

Among the companies that manufacture for telecommunications applications are several that also produce

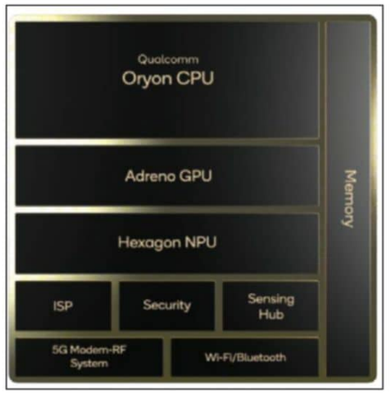


FIGURE 2
Qualcomm's Snapdragon X family of processors use the same CPU core in different numbers but vary in memory size and on-board accessory features. [Image courtesy Qualcomm]

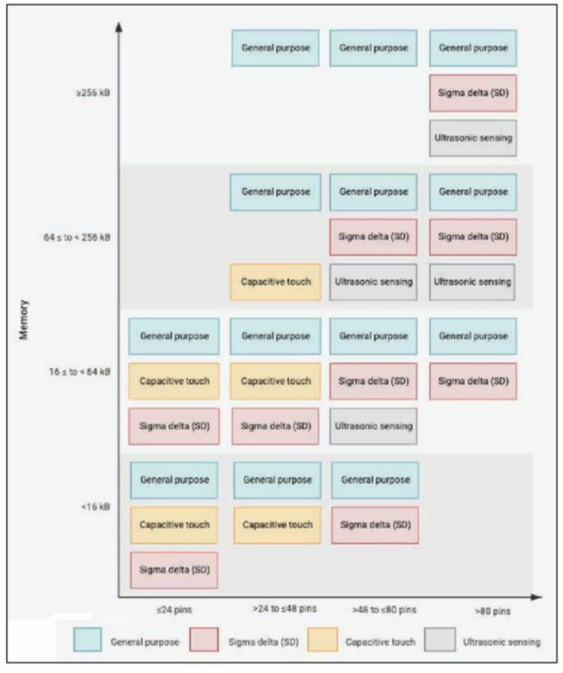


FIGURE 3
Texas Instruments' MSP430 portfolio includes more than 2,000 devices that vary widely in performance, auxiliary capabilities, and packaging. (Image courtesy Texas Instruments)

19843342

other components found in both endpoint and central office infrastructure.

Broadcom — A key player in networking and telecom ASICs, providing solutions for high-speed data processing [10].

Intel — Develops ASICs for telecom infrastructure, including data centers and networking applications [11].

Marvell Technology — Focuses on ASICs for high-speed networking and telecom infrastructure. Marvell's custom ASIC offering specializes in addressing the complex, silicon requirements of 5G carriers, cloud data centers, enterprises, and automotive applications [12].

MediaTek — Develops ASICs for mobile and broadband communication. MediaTek focuses a great deal of attention on 224G SerDes technology. A Serializer/Deserializer (SerDes) is a pair of functional blocks commonly used in high speed communications to compensate for limited input/output. FPGAs with built-in SerDes blocks make them ideal for critical applications including networking, high-speed transmission and reception, and critical video work [13].

REACHING AND MANAGING THE NETWORK

Telecommunications systems must effectively negotiate data moving over both physical and logical networks. In general the vendors that provide solutions for the logical networks and those that specialize in the physical networking layer are separate, though there are a handful that provide products in both.

Broadcom — A leading provider of networking and connectivity solutions, including Ethernet and fiber optic interface chips. Broadcom produces a wide variety of radio frequency, cabled, and optical interface chips and systems on a board [14].

Cisco Systems — A major player in networking hardware including network interface solutions for enterprise and telecom applications. Cisco network interfaces and switches are frequently found at the core of large-scale networks though the company does offer edge device connectivity, as well [15].

Marvell Technology — Provides advanced networking solutions, including Ethernet controllers and high-speed connectivity chips. Marvell network interface programs are targeted at large carriers, major corporations, and government units with high demand for their services [16].

MediaTek — Develops network interface chips for mobile and broadband applications. MediaTek interfaces are centered on the radio-frequency world, with a series of 5G and high-speed 802.11 adapters [17].

Intel — Known for its high-performance network interface controllers used in data centers and telecom infrastructure. Intel's network interface can support speeds up to 200GBe, as well as lower speeds for edge applications [18].

Qualcomm — Specializes in wireless communication chips, including 5G and Wi-Fi interface solutions. Qualcomm markets their network interfaces in tiers, for carriers, enterprises, and home networks. As the network tiers work up the scale in size, they add significant capabilities for managing subscribers and subscribed bandwidth, as well [19].

Realtek — Offers a range of network interface chips for consumer electronics and telecommunications. In addition, the company does provide network interfaces for both enterprise-class multi-network solutions and home or business end-point single network deployments [20].

BEYOND SIMPLE CAPABILITIES

When the requirement for radio-frequency networking goes beyond simple 802.11 capabilities, the hardware and its driving software become more specialized and more complex. Companies that provide commercial RF interface and support components tend to be larger and focus on more than the simple case of the end users and their laptop.

Broadcom — Supplies RF filters, amplifiers, and connectivity solutions for smartphones and telecom equipment.

Ericsson — Offers RF antennas and radio systems for global 5G deployments. The company promotes its radio access network (RAN) which is composed of hardware, software, and services to fully deploy a carrier-class communications network [21].

Huawei — Specializes in RF components for telecom infrastructure, including massive MIMO antenna technology. Huawei is a major global player in carrier-class networks globally, though their presence in North America has been constrained for the last few years. It remains to be seen whether Huawei will become an acceptable option in North America in the coming years [22].

Nokia — Develops RF transceivers and front-end modules for telecom networks. Nokia is one of a small handful of companies that is a realistic choice for customers buying at large carrier class capacity. Much of the company's capacity is focused at the network core [23].

Qualcomm — A leader in RF front-end solutions, providing modems and transceivers for 5G connectivity. Qualcomm is one of the rare companies that can drive technology and standard adoption around new protocols and standards, and it is already advancing capabilities for next-generation 5G network interfaces [24].

FIBER OPTIC INTERFACES

While wireless networking is responsible for more and more of the "last mile" of network availability, fiber optic interfaces are critical at the core and important in a growing number of end-point installations.

Accelink Technologies — Provides fiber optic access solutions, optical transceivers, and active optical cables. Accelink offers many of the basic interfaces and drivers required to populate larger core routers [25].

Broadcom (Avago) — Supplies optical transceivers and connectivity at a basic level. While Broadcom has many components that go into other products it also offers commodity transceivers for optical switches [26].

Cisco Acacia — Offers high-speed optical transceivers for data centers and telecom networks. Cisco provides not just the transceivers but the mainframe switches into which they go and every component along the optical infrastructure for telecommunications networking [27].

Additional materials from the author are available at: www.circuitcellar.com/article-materials
References [1] to [36] as marked in the article can be found there.

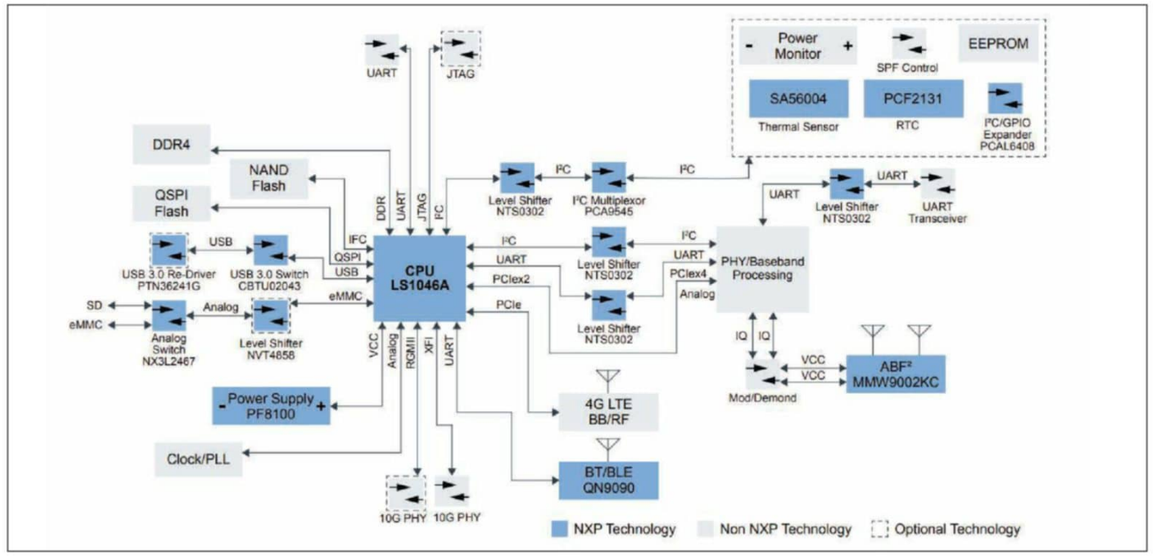


FIGURE 4
5G switches are complex, performance-critical systems that must also be high availability and feature rich. The CPU is supported by a fleet of support components for functions that range from RF transceiver to high-capacity multiplexer. (Image courtesy NXP)

Eoptolink — Develops high-speed optical modules for telecom and data communication. While Eoptolink is not as well known as some of the other vendors in this space, they provide transceivers with up to 800Gbps bandwidth for enterprise and carrier-class deployment [28].

NeoPhotonics (Lumentum) — Specializes in coherent optical modules and high-speed transceivers. Products are specialized in high bandwidth networks, including carrier-class deployments, under-sea cables, and very large enterprise networks [29].

Source Photonics — A global manufacturer of optical transceivers and optoelectronic devices, Source Photonics provides transceivers with bandwidth of up to 400Gbps. The company provides components to support many different network deployments, from fixed broadband to data center transmission, to wireless broadband [30].

FROM TRANSCEIVER TO PRODUCT

While the components that move bits or packets from one point to another are the critical foundation of telecommunications, they are not the entirety of the network. Whether the need is to manage a carrier network, have an endpoint device that a human can use to contact another human, or connect an IoT endpoint to a central point for command and control of an industrial process, other components and functions are required. Sensors and interface components make up many of these additional components, and they are sourced from a wide variety of companies.

Analog Devices — Specializes in high-performance analog and mixed-signal sensors used in networking and telecom equipment. Analog Devices also provide the interfaces that can connect users to the application and create the user experience that defines an application in many cases [31].

Broadcom — Offers optical and RF sensors used in telecom networks. The company has an extensive catalog of optical sensors for many application types, from environmental to industrial signaling [32].

Infineon Technologies — Supplies RF sensors and interface solutions for telecom and IoT applications. Infineon has a wide variety of sensors available for both fixed and mobile deployments, with application that runs the gamut from magnetic to pressure to gas presence [33].

NXP Semiconductors — Develops interface chips and sensors for wireless communication and networking (**Figure 4**), providing data on environmental and industry factors useful in both scientific and commercial applications [34].

STMicroelectronics — Provides sensors for environmental monitoring and interface solutions for telecom applications. STMicroelectronics is open about the importance it places on environmental sustainability and component longevity. This is carried across more than 50 types of sensors offered by the company [35].

Texas Instruments (TI) — The company provides scores of different sensor types and dozens of different interfaces for engineers to use in building applications in telecommunications. With a wide reach and a history of chips that brings with it product knowledge across the industry, sensors and interfaces from TI are common starting points for many engineers working to spec a solution [36].

CONCLUSION

The demands of application designers and customers will continue to drive telecommunications components toward higher performance and greater bandwidth. System manufacturers are already promising AI built into practically every telecom-enabled device, and independent designers will follow suit as AI-capable processors move down-market. Product engineers must never forget that the packets must flow, even with feature-rich temptation urging them to choose new options over traditional requirements.

DATASHEET

Datasheet:
AC-DC Power Supplies
More Power — Smaller Footprints

By
KC Prescott,
Publisher

The future of AC-DC power supplies forcasts smaller, more efficient, more adaptable, customizable, and seamless integration.

The evolution of AC-DC power supplies is rapidly reshaping the embedded systems landscape. Miniaturization and energy efficiency are no longer mere aspirations but critical design parameters. Switching power supplies, already dominant, are poised for even greater market share in 2025, exceeding 70%, thanks to their inherent advantages in efficiency, compact size, and wide input voltage tolerance. This dominance is further solidified by ongoing advancements in switching frequencies and sophisticated control techniques.

The relentless drive toward increased power density is a defining trend. Embedded systems are demanding more power within increasingly smaller footprints. This necessitates innovation across the board—from component miniaturization and advanced packaging to sophisticated thermal management solutions. The result will be power supplies capable of delivering significantly more power in remarkably smaller packages, enabling the creation of more powerful and compact embedded devices.

Energy efficiency remains paramount, especially in battery-powered applications. Improvements in efficiency directly translate to longer battery life and enhanced overall system performance. The industry is relentlessly pursuing higher efficiency through the implementation of advanced control algorithms, high-frequency switching, and the adoption of more efficient components such as GaN transistors. This pursuit is further fueled by increasingly stringent energy efficiency standards and regulations.

Customization and integration are also gaining traction. The demand for tailored power supply solutions is on the rise, particularly in specialized embedded applications. This trend is driving manufacturers to offer highly customizable power supplies, allowing engineers to precisely match voltage, current, and form factor requirements to their specific system needs. This results in optimized designs that maximize efficiency and performance.

Finally, the need for wide input range power supplies is becoming increasingly critical. The global deployment of embedded systems and the pervasive growth of IoT demand power supplies capable of operating across a broad spectrum of input voltages. This adaptability simplifies design, reduces complexity, and enhances the robustness of embedded systems in diverse geographical locations. The future of embedded systems hinges on the continued evolution of AC-DC power supplies. The trends described here paint a picture of a future where power supplies are smaller, more efficient, more adaptable, and more seamlessly integrated into the overall system design.

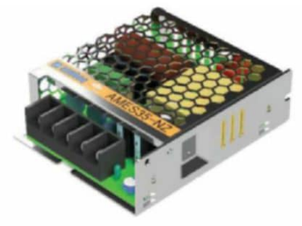
NEXT MONTH'S TOPIC: Single Board Computers Send related product announcements to editor@circuitcellar.com

Medical Use Applications

Advanced Energy's SL NGB150 series provide for a wide range of AC-DC embedded power requirements, offering voltages of 12V, 15V, 19V, 24V, and 48V together with an optional 5V standby and 12V fan output. Each model accepts a universal input of 85V AC to 264V AC. With full-load efficiency of greater than 90%, these compact switch-mode power supplies (SMPSs) feature output overvoltage and short-circuit protection.

- Output power: 150W max
- Dimensions: 4.0" x 2.0" x 1.26"
- Use: Industrial/Medical safety
- Temperature: -40°C to 70°C with derating
- 5V standby at 0.5A 12V fan output at 0.4A
- High efficiency: 90% typical
- ±10% adjustment range
- UL EN62368-1/UL60601
- CSA No. 62368-1/60601
- CB Report IEC62368-1, IEC60601-1
- EMI Class B
- Input range: 85V AC to 264V AC
- Frequency: 47Hz to 63Hz
- Output power: 120W
- Load regulation: ±2%
- Line regulation: ±0.5%
- Ripple and noise: 1% of Vout
- Over voltage protection
- Short circuit protection
- Overload protecion
- Overtemperature protection

Advanced Energy
www.advancedenergy.com



Cost-Effective Converter

The AMES35-NZ is an AC-DC converter designed for improved cost effectiveness, reliability, and performance. It accepts a universal input voltage ranging from 90V AC to 264V AC and 127V DC to 370V DC, providing output voltages from 5V to 48V with a maximum wattage of up to 38.4W. Notable features include a wide operating temperature range of -30°C to 70°C, high isolation voltage up to 4000V AC, and standard built-in output short circuit, over-current, and over-voltage protection. It is suitable for various applications including street lighting controls, industrial controls, and power grid systems.

- Designed for improved reliability and performance
- Dimensions: 3.89" x 3.22" x 1.18"
- Accepts a universal input voltage range of 90V AC to 264V AC/127V DC to 370V DC
- Output voltage range: 5V to 48V
- Maximum output: 38.4W
- Operating temperature range: -30°C to 70°C
- Isolation voltage: up to 4000V AC
- Output short circuit protection
- Output overcurrent protection
- Output overvoltage protection
- Low ripple and noise, typically up to 200mVp-p

Aimtec
www.aimtec.com

Low-Profile Design

The VGS-50E-12 boasts a rugged metal case and is certified to 62368-1 safety standards, ensuring reliable performance. Its has a low-profile design (34.5mm). Key features include over-voltage and short-circuit protection, along with Class B emissions compliance. The wide input voltage range and high efficiency (up to 88%) contribute to its versatility. DC input capability and universal input voltage further enhance its adaptability. Its operating temperature range of -30°C to 80°C (with derating).

- Metal case type
- Mounting: Chassis mount
- Profile: 34.5mm
- Overvoltage and short-circuit protection
- Class B emissions
- 62368-1 certified safety standard
- Input Voltage: Wide range, Universal
- Efficiency: 87%
- Operating temperature: -30°C to 80°C (with derating)
- Output power: 50W
- Isolation voltage: 4.25kV
- Output voltage: 12V DC
- Output current: 4.17A
- Overvoltage protection: 15.6V DC
- Ripple and noise: 120mVp-p
- Capacitive load: 4200µF

CUI, Inc. (a Bel company)
www.belfuse.com

DATASHEET URLS:

Advanced Energy: www.advancedenergy.com/getmedia/428279c2-2d4c-4d1d-9fd3-fc94cfdbf0d9/eng-ngb150-235-01-01-22-24.pdf

Aimtec: https://aimtec.com/site/Aimtec/files/Datasheet/HighResolution/ames35-nz.pdf

CUI: www.belfuse.com/media/datasheets/products/power-supplies/VGS-50E.pdf

DATASHEET

Datasheet:
AC-DC Power Supplies

Programmable DC Power

The Magna-Power SL Series stands out as a high-performance, compact, and versatile family of programmable DC power supplies. Engineers appreciate its high power density, enabling efficient use of lab space. The series' advanced digital control ensures precise voltage and current regulation, minimizing noise and ripple. Its robust design includes comprehensive protection features, safeguarding both the supply and the connected load. The intuitive interface and extensive programmability via SCPI commands simplify integration into automated test systems. The SL Series is a reliable and adaptable solution for a wide range of applications, from component testing to system integration. Its combination of power, precision, and ease of use makes it a valuable asset in any engineering environment.

- Output power: 1500W to 10000W
- Output ripple and noise: <1mV
- Voltage resolution: 1mV
- Current resolution: 1mA
- Line regulation: <0.01%
- Load regulation: <0.01%
- Transient response: <50µs
- Operating temperature: 0°C to 40°C
- Dimensions: 1U Rack Mount
- Efficiency: >85%
- Protection features: overvoltage, overcurrent, overtemperature
- Cooling: Fan-cooled
- Display: Digital
- Interface: RS-232, USB, LAN, SCPI
- Input Voltage: 208V AC/230V AC

Magna-Power
www.magna-power.com

All-In-One Power Supply

The DRS-480-24 integrates several crucial functions into a small footprint, saving valuable space in installations. Its key distinction lies in combining AC-DC power supply, battery charging, DC-UPS functionality, and digital communication (Modbus) in a single unit. This all-in-one design simplifies system integration, reduces component count, and minimizes wiring complexity. High efficiency and wide operating temperature range further enhance its appeal for demanding applications. Robust protection features such as overload, overvoltage, and battery reverse polarity safeguards ensure reliable operation. The device is particularly well-suited for security systems, industrial automation, and other applications requiring reliable power and backup.

- Input voltage: Universal
- Output voltage: 24V DC
- Output current: 20A
- 480W output power
- High efficiency up to 93.5%
- Operating temperature: -30°C to +70°C with free air convection
- DIN-rail mounting
- Metal enclosure
- Battery charging capability
- DC-UPS functionality
- Modbus communication
- Overload protection
- Overvoltage protection
- Battery low voltage disconnect
- Battery reverse polarity protection
- PFC function

Mean Well
www.meanwell-web.com

For Demanding Applications

The Ominion Power HPR2812DC is a high-performance DC power supply designed for demanding applications, featuring a compact size with a power density of 68.6W/in³. It operates within a DC input range of 36V DC to 75V DC, making it versatile for various setups. With an impressive MTBF of over 1.9 million hours, it ensures reliability and longevity. The unit is convection-cooled and offers high efficiency, making it suitable for both industrial and outdoor environments. Its robust design and advanced features make it an excellent choice for engineers seeking dependable power solutions.

- Output power: 3000W
- Input voltage range: 36V DC to 75V DC
- Efficiency greater than 90%
- MTBF over 1.9 million hours
- Power density: 68.6W/in³
- Convection cooling method
- Compact design
- Lightweight
- Protection features overvoltage, overcurrent, short-circuit
- Operating temperature: -40°C to +70°C
- Output voltage: 12V DC
- Output current: 60A
- Meets relevant safety standards
- Ingress Protection IP65 rated
- Control Interface: Remote monitoring capability
- Versatile mounting configurations
- Low noise operation

OmniOn Power
www.omnionpower.com

Offers Industrial Reliability

The RECOM RACPRO1 DIN rail power supply stands out for its exceptional power density and compact design, ideal for space-constrained applications. Its wide input voltage range and high efficiency minimize energy loss, while advanced protection features ensure reliable operation. The RACPRO1's robust construction and high-quality components guarantee long-term durability. Engineers will appreciate its ease of integration and compliance with industry standards, making it a versatile choice for various industrial and automation projects. Its innovative thermal management ensures optimal performance even under demanding conditions.

- Input voltage range: 85V AC to 264V AC
- Output voltage options: 5V, 12V, 24V, 48V
- Efficiency: up to 94%
- High power density
- Operating temperature: -25°C to +70°C
- Protection features: short-circuit, overload, overvoltage
- Mounting DIN rail
- Compact dimensions
- Isolation 3kV AC
- EMC compliance meets relevant standards
- Safety approvals: UL, cUL, IEC/EN 62368-1
- No-load power consumption
- High MTBF
- High input surge protection
- Low ripple and noise
- Low THD

RECOM
www.recom-power.com

Space-Saving Power Supply

The D1SE120-24-A3 by TDK is a compact and efficient AC-DC power supply designed for DIN rail mounting, making it ideal for space-constrained applications. With a rated output of 24V DC and an efficiency of up to 92%, it minimizes energy consumption and heat generation. This model supports both AC and DC input, providing flexibility for various applications. Its robust design includes comprehensive protection features, ensuring reliable operation in demanding environments. The D1SE120-24-A3 is an excellent choice for industrial and medical applications requiring high performance and reliability.

- Input voltage range AC: 100V AC to 240V AC
- Input voltage range DC: 110V DC to 250V DC
- Rated output voltage: 24V DC
- Output voltage range: 22.5V DC to 29V DC
- cCurrent: 0 to 5A
- Efficiency: up to 92%
- Operating temperature: -20°C to +70°C
- Compact design
- Weight maximum: 460g
- Protection features: Overvoltage, overcurrent, short-circuit, overtemperature
- Cooling method: convection or forced air
- Mounting type: DIN rail
- Ingress protection: IP20
- Class of protection: Class I
- Hold-up time: minimum 23ms

TDK-Lambda Germany
www.tdk.com

Noise-Sensitive Medical Grade

The CCR200 series features compact medical AC-DC power supplies that function effectively in convection, conduction, and fan-cooled settings, with power ratings ranging from 110W to 200W. Designed for high efficiency and power density, it utilizes a low-profile, baseplate-cooled structure that optimizes thermal management while ensuring quiet operation for sensitive environments. Compliant with Class B emissions and equipped with global safety certifications, 2x MOPP isolation, and minimal leakage currents, the CCR200 is suitable for a variety of BF-rated medical applications, such as respiratory care, imaging, patient monitoring, and industrial tasks.

- 200W fan cooled, 170W conduction cooled
- Universal, single phase input 85V AC to 264V AC
- Dimentions: 78.6mm x 57.9mm footprint, 25.4mm profile
- High efficiency: up to 94%
- Low earth leakage: <300µA fLow patient leakage <75µA
- Medical (BF) & ITE safety approvals
- Class B conducted and radiated emissions
- Operating temperature ranges: -40°C to +80°C, with safety approved up to 50°C
- Storage temperature: ranges from -40°C to +85 °C
- Weight: approximately 220g (0.485lb)
- Overload protection: between 125% and 165% of the nominal current

XP Power
www.xppower.com

DATASHEET URLS:

Magna-Power: https://magna-power.com/assets/files/datasheets/datasheet_sl_4.0.5.pdf

Mean Well: www.meanwell-web.com/en-gb/ac-dc-industrial-din-rail-power-supply-output-drs--480--24

OmniOn Power: www.omnionpower.com/assets/pdfs/windchill/data-sheet/hpr2812dc_ds.pdf

DATASHEET URLS:

RECOM: https://recom-power.com/en/products/ac-dc-power-supplies/ac-dc-din-rail/rec-c-ac-dc-din-rail.html

TDK-Lambda: https://www.emea.lambda.tdk.com/de-en/landing/d1se

XP Power: www.xppower.com/portals/0/pdfs/SF_CCR200.pdf

Modern Embedded Insights

Improve Portability and Reusability in Embedded Systems with West

Discover how West, a meta-tool from the Zephyr real-time operating system (RTOS) ecosystem, revolutionizes embedded software development by streamlining project management across multiple repositories. This article dives into practical applications of West for enhancing portability and reusability in complex embedded systems, offering engineers actionable insights to optimize their workflows.

By
Jacob Beningo

Embedded software development is rapidly evolving. What once was a bare-metal monolith compiled in a vendor integrated development environment (IDE), is now transitioning into highly modular, reusable, and testable codebases driven by modern DevOps workflows.

In previous articles, we explored how to build a modern CMake build system [1] for embedded development and how to apply Test-Driven Development (TDD) techniques effectively in that environment.

Today, we will explore another game-changing tool: West, a meta-tool from the Zephyr RTOS ecosystem [2]. Even if you don't use Zephyr, West offers several valuable lessons for managing complex embedded projects with many components, targets, and repositories.

In this article, we'll walk through:

- What West is and why it matters.
- How Zephyr uses West to manage code at scale.
- How to use West in your own embedded projects.
- How it complements a modern build system.
- Practical examples to demonstrate reusable, portable architecture with West.

Let's dive in.

WHAT IS WEST?

At its core, West is a project and workspace management tool. Originally developed as part of the Zephyr RTOS project, it was designed to solve a recurring pain point in embedded development—managing multiple Git repositories across layers of dependencies.

You can think of West as a wrapper for Git, CMake, and custom build commands that makes it easy to:

- Clone and manage multiple repositories in a workspace.
- Build complex projects with multiple targets or boards.
- Run test, flash, or debug commands with consistent behavior.
- Define and manage extensions or subprojects in a reproducible way.

While Zephyr is the canonical user of West, nothing stops you from using it to manage your own RTOS, middleware, applications, and even continuous integration (CI) workflows.

WHY YOU SHOULD CARE (EVEN IF YOU DON'T USE ZEPHYR)

Embedded development is increasingly modular. In fact, as you look at more and more devices being connected to the Internet, there are plenty of opportunities for reuse.

For example, your current projects might already be pulling in:

- Board support packages (BSPs) from silicon vendors.
- RTOS source trees (FreeRTOS, Zephyr, etc.)
- Third-party middleware such as MQTT, BLE stacks, or file systems.
- Custom reusable libraries across multiple products.

Managing all of this by hand is a nightmare. Git submodules are finicky. Build configurations drift. Manual integration becomes fragile. West solves these problems by:

- Defining project manifests in a single YAML file.
- Providing commands to fetch, sync, and build projects.
- Offering per-project overrides for build flags, remotes, and directories.

Let's look at how Zephyr uses West to scale. Then, we'll walk through adapting it for your own systems.

HOW ZEPHYR USES WEST (AND WHAT WE CAN LEARN FROM IT)

If you've spent any time exploring Zephyr RTOS, you know it's a beast. We're talking about a massive project that supports hundreds of boards, SoCs, and modules. It's impressive, but also intimidating if you're coming from a more traditional embedded workflow.

So how does Zephyr keep it all straight? The answer is West.

Zephyr uses West to manage its sprawling codebase and keep development sane. And even if you never plan to use Zephyr in a product, there are some key lessons you can apply to your own projects.

First, Zephyr is broken into dozens of repositories. Here are just a few of the major components:

- `zephyr/` — the kernel and base operating system
- `hal_stm32/`, `hal_nordic/`, etc. — hardware abstraction layers for different vendors
- `cmsis/`, `mcuboot/` — peripheral libraries and bootloaders
- `modules/` – networking stacks, middleware, and more

If you tried to clone and maintain these manually, it would be a nightmare. Git submodules? Good luck. Manual scripts? Been there, done that—never again.

Instead, Zephyr uses a single west.yml manifest to define all its repositories and where they belong in the workspace. The manifest looks something similar to **Figure 1**.

Once that manifest is in place, you run two simple commands:

```
west init
west update
```

...and just like that, your entire workspace is populated with the exact versions of each repository. There are no mismatched commits. No searching GitHub for the right tag. Everything lands precisely where it should, ready to build.

It's like infrastructure-as-code, but for your entire firmware stack.

Second, Zephyr uses West to build, flash, and debug. For example, if you wanted to build your project for the STM32F446RE Nucleo board, you'd use a command such as:

```
west build -b nucleo_f446re path/to/app
```

Once the project is compiled, you can use the following command to flash the code to your target with:

```
west flash
```

That command will just program the application to the target. It won't let you debug it. If you want to do that, you need to use the following command:

```
west debug
```

As simple as these two examples may seem, they standardize the entire workflow—from code retrieval to deployment—so you can focus on developing features instead of configuring tools.

```
manifest:
  projects:
    - name: zephyr
      url: https://github.com/zephyrproject-rtos/zephyr
      revision: main
    - name: hal_stm32
      url: https://github.com/zephyrproject-rtos/hal_stm32
      path: modules/hal/stm32
```

FIGURE 1
Zephyr uses a single `west.yml` manifest to define all its repositories and where they belong in the workspace.

GETTING STARTED WITH WEST IN YOUR OWN PROJECTS

So now that we've seen how Zephyr uses West to manage a massive project, you might be thinking, "That's great for Zephyr. But I don't have a giant RTOS project with 50 repositories. Can I still use West in my own stuff?"

Absolutely.

In fact, West can be even more powerful in smaller projects because it helps you organize your codebase before it grows out of control. Whether you're managing reusable drivers, different applications, or just want a cleaner dev setup, West gives you a consistent, reproducible way to get the job done.

Let's walk through how you can use it in your own embedded development workflow.

Step 1—Set Up a Manifest Repository: Every West-based project starts with a manifest. Think of this as the blueprint for your workspace. It's where you define what repositories you're using, where they live, and what version of each one should be checked out.

Let's say you've got a setup such as the exmaple shown in **Figure 2**. Instead of manually cloning each repo or setting up fragile relative paths in your build system, you can define everything in a `west.yml` file, as seen in **Figure 3**.

Once the manifest is ready, you set up the workspace like this:

```
west init -m https://github.com/
myorg/my_workspace_manifest
west update
```

That's it. All your components are pulled into the right place, at the correct version, without any guesswork. You've just leveled up your repo hygiene.

Step 2—Integrate with Your CMake Build System: West doesn't replace your build system, it complements it. You can use the same modern CMake setup we've previously discussed. Just organize your projects using CMakeLists.txt files and point to your shared libraries using paths that West sets up automatically.

Here's an example CMakeLists.txt for your `sensor_node` app:

```
cmake_minimum_required(VERSION
3.20)
project(sensor_node)
add_subdirectory(${WEST_ROOT}/
common_libs/drivers)
add_executable(sensor_node main.c)
```

Now you can build your app like this:

```
west build -b custom_board . --
-DCMAKE_TOOLCHAIN_FILE=toolchain-
host.cmake
```

No more messing with include paths, copying driver code between apps, or hardcoding toolchain settings. It all flows through one system.

Step 3—Create a Developer Workflow that Just Works: You don't need to memorize long build commands or create a dozen different shell scripts. West lets you streamline your workflow with clean, consistent commands for common tasks.

Need to test something?
Need to rebuild and flash firmware?
Need to debug over SWD?

You can customize your workflow using simple commands that your whole team can follow. And if you really want to go deep, you can even add custom West extensions for things such as unit tests, static analysis, or CI automation. We'll talk more about that in the next section.

Step 4—Extend West with Custom Commands: One of the coolest things about West and, something that most developers overlook, is that it's not just for pulling repos and building code. You can actually extend it with your own commands.

That means you're no longer tied to a patchwork of shell scripts, Make targets, or README instructions. You can give your team a consistent, discoverable interface for common tasks, just like Zephyr does.

Let's walk through what that looks like.

WHY ADD CUSTOM COMMANDS?

If you've ever:

- Written a `run_tests.sh` script for unit tests.
- Created a dozen README instructions on "how to build for board X."
- Had to explain to your coworker (again) how to flash the firmware.

...then a custom West command is probably the better way.

By defining these commands once, you standardize the workflow and make it easy for anyone on your team (or future you) to do the right thing with a single, simple command.

CREATING YOUR FIRST CUSTOM COMMAND

Let's say you want to add a West test command that runs your unit tests using `ctest`.

Here's how to do it:

1) Create a directory in your repo (or workspace) called west_commands/.
2) Inside it, create a Python file called test.py with the content shown in **Figure 4**.
3) Update your west.yml to register the custom command seen in **Figure 5**.

Once that's set up, you can run your tests like this:

```
west test
```

That's it. You've just replaced an entire section of your README with a real command that anyone can run anywhere, anytime.

```
my_workspace/
├── apps/
│   ├── thermostat/
│   └── sensor_node/
├── common_libs/
│   ├── drivers/
│   └── utils/
├── rtoses/
│   ├── freertos/
│   └── embos/
```

FIGURE 2
An example setup.

```
manifest:
  projects:
    - name: apps
      path: apps
      revision: main
      url: https://github.com/myorg/embedded-apps
    - name: common_libs
      path: common_libs
      revision: main
      url: https://github.com/myorg/embedded-libs
    - name: freertos
      path: rtoses/freertos
      revision: main
      url: https://github.com/myorg/freertos-port
```

FIGURE 3
You can define everything in a `west.yml` file.

```
from west.commands import WestCommand

class Test(WestCommand):
    def __init__(self):
        super().__init__(
            'test',
            'Run unit tests using CTest',
        )

    def do_run(self, args, unknown_args):
        import subprocess
        subprocess.run(['ctest', '--output-on-failure'], check=True)
```

FIGURE 4
Create a Python file called `test.py` in the west_commands/ directory in your repo.

```
self:
  west-commands: west_commands
```

FIGURE 5
Sample workspace organization.

The power here isn't just in automation, it's in standardization. Everyone runs the same commands. Everyone gets the same results. And no one has to guess how things are supposed to work.

That's how you scale embedded software development without losing your sanity.

FIGURE 6
An example of an organized workspace.

```
west_workspace/
├── apps/sensor_node/
│   ├── main.c
│   └── CMakeLists.txt
├── common_libs/
│   └── drivers/i2c_sensor/
│       ├── i2c_sensor.c
│       └── i2c_sensor.h
├── hal/
│   └── stm32/
│       ├── gpio.c
│       ├── spi.c
│       └── CMakeLists.txt
└── west.yml
```

REUSABILITY AND PORTABILITY: LESSONS FROM ZEPHYR'S ARCHITECTURE

If there's one thing we should all learn from Zephyr, it's this—separate your application from your platform.

Zephyr's architecture enables thousands of configuration options and forces you to think modularly. Drivers, applications, board support packages, and libraries are all broken into separate components, each with their own purpose and well-defined boundaries.

And that mindset? It's one we should all adopt whether or not we use Zephyr.

Let's unpack why.

Most embedded developers (my past self included) start by jamming everything into a single folder: application code, peripheral drivers, glue logic, and maybe a FreeRTOS folder tossed in for good measure.

It works—until it doesn't.

Suddenly, you need to:

- Port your app to a new microcontroller.
- Reuse a driver in a different product.
- Run a unit test without bringing up hardware.

And now you're untangling a web of tight coupling that you never meant to build.

That's where West—and the architectural philosophy behind it—comes in.

By organizing your code into standalone, versioned components and managing them with a manifest, you make reuse and portability a built-in feature, not an afterthought.

Here's how you can set yourself up for success using West to help you architect your project for reuse:

- Break out drivers and middleware into their own repos (or at least directories).
- Use west.yml to pull in only the components you need for a given application.
- Swap hardware layers by changing a manifest entry, not by rewriting your app.
- Target different boards or platforms by passing `new toolchain` or `build config` arguments to `west build`

Want to reuse your thermostat control app across an STM32 and an NXP processor? You shouldn't have to rewrite a single line of application code. Just update your HAL layer in the manifest and switch toolchains.

Done.

Want to share a driver with another team? No problem. Package it, point to it in your manifest, and use it just like any other dependency.

```
manifest:
  projects:
    - name: sensor_node
      path: apps/sensor_node
    - name: i2c_sensor
      path: common_libs/drivers/i2c_sensor
    - name: hal_stm32
      path: hal/stm32
```

FIGURE 7
A trimmed-down version of a manifest, where each component is defined as a separate project.

Additional materials from the author are available at:
www.circuitcellar.com/article-materials
References [1] to [2] as marked in the article can be found there.

When everything is modular and version-controlled through a manifest, it also becomes easier to:

- Reproduce builds across environments.
- Run tests on subsets of components.
- Automate builds for multiple targets in parallel.

And let's be honest, there's no better feeling than knowing your system builds cleanly every time, from a fresh clone, on any machine.

It's the difference between *"Works on my laptop"* and *"This thing's rock solid."*

PRACTICAL EXAMPLE—BUILDING A SENSOR NODE WITH A REUSABLE HAL

Let's pull everything together with a real-world example.

Imagine you're building a low-power sensor node. It's going to ship on an STM32 platform, but you're already planning to migrate to a different processor family next year—maybe NXP or Renesas. You want to keep your application logic untouched and simply swap out the hardware layer when needed.

This is exactly the kind of setup where West shines.

You might organize your workspace as shown in **Figure 6**. Your application is clean. It has no idea which vendor HAL is being used under the hood. That's the way it should be.

In your manifest, you define each component as a separate project. A trimmed-down version can be seen in **Figure 7**.

Need to switch from STM32 to an NXP platform? Just change hal_stm32 to hal_nxp and re-run `west update`. You don't touch your app. You can use a single command to build the application:

```
west build -b custom_board apps/sensor_node
```

To run your tests, you can use:

```
west test
```

FINAL THOUGHTS

If you've been following along, you've now seen how West offers a clean, scalable, and efficient way to manage embedded software projects.

It doesn't matter whether you're working on a simple sensor node or a complex platform with multiple products. West gives you the structure, tools, and repeatability needed to build modern, maintainable systems.

At this point, you've got the tools to:

- Organize your project into modular components.
- Reuse code across platforms and products
- Separate hardware-specific logic from your core application.
- Simplify and standardize your build, test, and deployment workflows.

The key takeaway here is simple—West isn't just for Zephyr. It's a powerful tool you can adopt incrementally, starting with just a manifest and a few repositories. As your needs grow, West scales with you. You don't need to change your entire development process overnight. You just need to take the first step.

Start by:

- Creating a basic west.yml manifest that defines your project structure.
- Refactoring one of your applications to separate the HAL from the core logic.
- Building it using West and your existing CMake setup.
- Running it on different targets with minimal changes to the application.

The sooner you start thinking in terms of portability and reusability, the easier your life becomes down the road. With a proper build foundation in place, you're well-positioned to bring in other modern techniques such as simulation, DevOps, and continuous integration without rebuilding your toolchain every time. These practices not only streamline workflows but also enhance collaboration across teams, saving valuable time and resources.

Getting started doesn't take much. But once you do, you'll never look at embedded project organization the same way again. By adopting these principles early, you set yourself up for long-term success and adaptability in an ever-evolving technological landscape.

ABOUT THE AUTHOR

Jacob Beningo is an embedded software consultant and founder of the Embedded Software Academy, where he helps engineers master modern practices to design efficient, scalable embedded systems. He helps developers stay ahead in an evolving industry with hands-on training, expert insights, and strategies. He's consulted and trained companies such as Arm, General Motors, NASA, and others, completing more than 200 projects. Jacob has published books on embedded software development techniques. He holds three degrees, including a master's in space systems engineering from the University of Michigan. You can learn more about Jacob's work at www.beningo.com.

COLUMNS

Embedded in Thin Slices

An Embedded Designer Looks at Mobile App Development

Releasing Your Flutter App to the App Stores

This month, Bob wraps up his article series on mobile app development from the perspective of an embedded systems designer. He looks at what it takes to release your Flutter app to the Apple and Google app stores, and then gives a high-level, opinionated, overview of the two paradigms—Flutter and React Native.

By
Bob Japenga

As I have discussed in previous articles in this series, I developed my React Native app using Expo. I found it easy to use with very few downsides. There were the normal quirks that we get from all software (I can't believe I just said that!), with things not always working the same and the constant changes made underneath us that are maddening. But software developers like us have conditioned the public to expect that. It gives us a "get out of jail free" card for our lack of planning, design, and testing.

Expo provided Expo Application Services (EAS), which supplies services to simplify all aspects of alpha, beta, and production releases. I expected there to be something similar for Flutter for releasing the app to the app stores. Instead, I found a documentation page [1] that tells you how to release for Apple's App Store and Google's Play Store. This documentation includes all kinds of esoteric files that I never had to deal while releasing with EAS. And the procedures between the two platforms are very different (**Figure 1**).

When I complained to a friend, a long time Flutter developer, he recommended that I look into CodeMagic [2] (**Figure 2**) When I created my account with them, they said that with CodeMagic you could automate the most time-consuming app release steps, such as code signing, building and uploading new app versions, and setting changelogs.

All of this is not time-consuming with EAS, so clearly the free market has filled the need already for something the Flutter ecosystem doesn't provide. Of course, when I tried CodeMagic for the first time, it hung up; and the webpage where it was hung said, "Click here to try again," if it hung up.

I didn't even complain. I just clicked there. And now it hung up in a different state! The next time, through, it seemed to work. See what I mean about how we have trained the public?

Of course, CodeMagic comes at a cost, but "time-consuming" steps scared me away. And the cost for a personal account is not prohibitive. What do they say—"Time is money?" As of 2025, the cost is $50 per year. Plus there is a free level. It was definitely better than what Flutter provided, but a far cry from Expo EAS.

PRELIMINARY PLANNING

Before you release your app to the app stores, you need to make some decisions. Do you need to prevent others from reverse-engineering your code? What do you want to use as an icon for your app? Do you have a developer account with both Google and Apple? Is the app a free or a purchased app? In what countries do I want to distribute my app? Am I going to monetize the app? Will I record user data (even username and password)? Am I using encryption techniques that cannot be exported? You will need answers to these kinds of questions when you register with Google or Apple.

OBFUSCATION

I remember when I first heard the word "obfuscate" and I thought it was a joke. Basically it is a big word for "hide." Reverse-engineering tools have come a long way since I first tried to reverse engineer some of the code in "Where in the World is Carmen Sandiego?" (Shhh! It did violate the license agreement!) There are tools out there on GitHub that will take you a long way toward reverse engineering Dart/Flutter code.

Flutter provides a way to obfuscate the code, but I don't see how it would significantly slow down a serious team bent on reverse engineering your code. However, it will make their job harder by making all variable names, class names, and function names generic. If this is important to you, you need only add the "—obfuscate" switch to the "Flutter build" command. But some obfuscated code may actually fail. Buyer beware.

A better way to slow them down might be to incorporate some native code in your mix (Java or Kotlin for Android, or Swift or Objective-C for iOs).

FLAVORS

I created a React Native app that is used in several different environments. Let me mention three. All of them use all of same source code.

The first flavor is called the Sermon Engagement app. Each week someone from the church takes the sermon and the sermon's slides, and creates from one to 10 engagements. They take a particular point from the sermon, and summarize it and provide questions to probe deeper into what was said. Some slides are included to help reinforce the message. References to links are provided to material that was either handed out or referenced. All of the weekly data is stored in the Cloud.

A second flavor is used by a book author to promote his book [3]. The book has more than 70 paths or exercises intended to help us grow in humility. This flavor includes a dozen or so of the exercises, and then includes a link to buy the book.

A third flavor was a Christmas present I made for my grandson who got his pilot's license. I took a book about flying (whose copyright had expired) and made it into a book he could both read—with pictures and maps from the book and elsewhere—and listen to his grandpa and grandma read.

Flutter provides a similar feature [4]. Whereas with React Native and Expo, a single

FIGURE 1
React Native vs. Flutter.

method is used for creating flavors, Flutter requires separate procedures for Android and iOS. This is a somewhat annoying trait of the Flutter ecosystem that frankly is just frustrating and time consuming. Every developer must wade through the different approaches of Android and iOS, rather than a single developer creating a wrapper that handles both.

RELEASING TO THE APP STORES

Once the preliminaries are out of the way, Flutter documentation provides you with a detailed, step-by-step procedure for releasing to Android [5] and to iOS [6]. Again, keep in mind "time consuming," and that it will require you to dip into confusing details about Android and iOS that I didn't need to know with Expo EAS. With EAS, I released to both app stores in hours. With Flutter, it took me days.

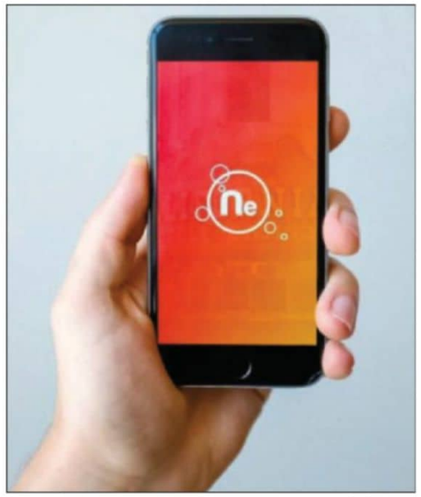
A BRIEF SUBJECTIVE COMPARISON

As I wrap up this series, I am going to provide a subjective comparison of the two ecosystems—React Native with Expo using JavaScript, and Flutter using Dart. I have read a lot of the comparisons on the Web, and they all say pretty much the same thing. But it is not clear that the authors have actually developed using both. I am not going to give facts and figures about app size, or speed of execution, or time to load, or maturity of the tools and widgets. What I will provide is my gut reaction after fully developing and releasing an app that runs on Android and iOS with both React Native and Flutter.

FIGURE 2
CodeMagic logo.

COLUMNS

FIGURE 3
A generic example of the Flutter Splash Screen.



Programming Language: JavaScript vs. Dart: In my very first article in this series [7], I mentioned that I chose React Native over Flutter because of Dart. I did not want to learn a new language. I already "knew" JavaScript and didn't like it but I still was familiar with it and its many libraries. Had I looked into Dart, I would have seen that it combined features of languages I already knew (C and JavaScript), so this should not have been a big deciding factor. Overall, I like Dart. Its strong typing can be maddening, but I don't hold it against that, because I know it is good for me—for the most part.

Score: JavaScript 3, Dart 7

Ease Of Development: Expo vs. Bare Metal Flutter: For my React Native app, I used Expo (called "Managed Workflow"), rather than what they called "Bare Metal" using the React Native Command Line Interface (CLI). For Flutter, there were no other options such as Expo. I had zero downsides using Expo. But designing and coding my app in Flutter took much longer and was much more work, primarily because it was more like Bare Metal React Native.

For testing and debugging, the idea of having a separate app (Expo Go) on your phone made connecting the development tools much simpler either with Wi-Fi or USB. Every time I went away for a day from a Flutter debug session and then came back, it was an adventure to connect my phone over Wi-Fi or USB. Of course, the gigantic drawback for Flutter is that I had to have a Mac to debug and test an iOS Flutter app. My Linux or Windows development machines won't do it. Yuck.

Score: Expo 8, Flutter 4

Ease of Release Deployment: If you are not a Mac person, Flutter has a huge downside. You must have Xcode to do a release. For a Windows/Linux user, the Mac is just not intuitive, no matter what people say. I have had a much harder time navigating my Mac than I did when I first used Linux or OS/2. I was glad to get a confirmation of this from my granddaughter the other day. She is not any kind of a computer geek and has had an iPhone since her teens. But when she switched jobs at her workplace and was issued a MacBook instead of a Windows laptop (different IT departments!), she echoed the same concerns I had.

Simple tasks become major efforts until you figure out the quirks of the user interface. Start with X'ing out a window. The stoplight analogy is just not intuitive! It's not that Windows or Linux are intuitive, but we know the quirks!

So, because of its requirements to have a Mac for release, and that Flutter has two completely different procedures for releasing to the Play Store and the App Store, the Expo score is much higher.

Score: Expo 9, Flutter 2

Splash Screen: Expo made creating a splash screen a breeze by being built into the ecosystem. Flutter required a very buggy plug-in to have a single path to a splash screen, without having to plunge into the depths of the archaic iOS and Android methods of this required feature. A generic splash screen is shown in **Figure 3.**

Score: Expo 9, Flutter 5

Code Stability: I had code stability problems with both ecosystems. The code underneath was frequently changing. Going from one version of the SDK to the next created time consuming problems with both. Once, when I went from Expo SDK version 49 to SDK version 50, I got errors in my build. I spent the better part of a day trying to find the cause. Then I updated to SDK version 51, and all problems magically went away.

Take a look at **Figure 4.** Do things stay centered with a small perturbation (Stability a)? Or do things move a little with a small change (Stability b)? Or do things go flying if the wind picks up just a little (Stability c)?

Score: Expo 4, Flutter 4
(both are between stability types b and c)

Flutter Widgets vs. React Native Components: This comparison comes down to the difference between Components (React Native) and Widgets (Flutter). I found that the Flutter widgets were more intuitive to create object-oriented designs with inheritance. In other words, creating a base class or Widget, and then building more complex widgets inheriting the children's attributes is easier. My mind easily adapted my C++ experience with the Flutter paradigm. After a time, I found myself in design thinking about widgets. That did not happen with React Native and components. Overall, I found I was starting to think Flutter and Dart. This did not happen with React Native.

Score: Flutter 7, React Native 4

Stable Libraries/Packages: Here is my non-qualitative comparison. It doesn't matter who has more widgets or components. What matters is: Do they work? Are they easy to install and use? Is the documentation sufficient. Here I rate both as a 10. Tons of widgets and components to choose from.

Asynchronous Programming: This was covered for each ecosystem in my April 2024 [7] and December 2024 [8] articles, but the bottom line is that the Dart Future is better than the React Native JavaScript Promise. The tools to handle asynchronous programming seem to be more deterministic and robust in Flutter. Plus I still have an unsolved asynchronous bug in my React Native code that does not exist in my Flutter code. That said, it seems as if the Flutter paradigm has too many options, which adds to our learning curve.

Score: Flutter 6, React Native 4

State Management: The ecosystem for each state management solution was covered in my October 2024 [9] and April 2025 [10] articles in *Circuit Cellar.* The presence of many more third-party, state management solutions in Flutter indicates the clear winner here. React Native's solution is less powerful but simpler and more robust. They both could use work, however.

Score: Flutter 2, React Native 5

COMPARISON SUMMARY

My ratings of the various aspects of Flutter and React Native are given in **Table 1**, using a scale of 1 (poor) to 10 (excellent).

The purist in me likes Flutter and Dart, with which you can create some elegant designs. I am not talking about the elegance of the user interface, but rather, of the underlying code structure. The code is neat and clean. The design can be made stable and reusable. But the release and development tools for Flutter are so painful, that knowing what I know now, I would still choose React Native with Expo over Flutter. The development time was much quicker even if the code was not as elegant.

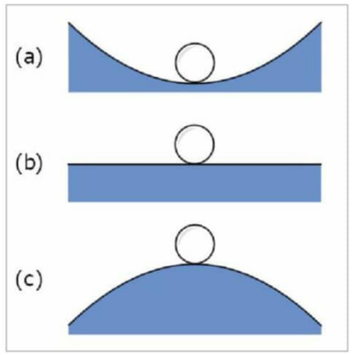


FIGURE 4
Types of code stability.

Task	Rating	
	React Native/Expo	Flutter
Programming Language	3	7
Ease of Development	8	4
Ease of Release/Deployment	9	2
Splash Screen	9	5
Code Stability	4	4
Flutter Widgets vs. React Native Components	4	7
Stable Libraries/Packages	10	10
Asynchronous Programming	4	6
State Management	5	2

TABLE 1
My ratings of various aspects of React Native/Expo and Flutter, on a scale of 1 (poor) to 10 (excellent).

CONCLUSION

Releasing your mobile app to the app stores is much harder with Flutter than it was with React Native and Expo EAS. I suspect that this is a maturity issue with Flutter and I would expect it to become easier within the decade. For my entire career, I always wanted to do two complete projects with completely different paradigms to compare them. Acomparative analysis broadens our technical skills and enhances our understanding of the strengths and weaknesses inherent in each framework. This was never possible as for-profit developers. But as a retired embedded engineer, it was a fun and mind-expanding exercise. We learn a lot when we see in depth two totally different approaches to the same problem. Even when you are doing it in thin slices.

ABOUT THE AUTHOR

Bob Japenga has designed embedded systems since 1973. From 1988-2020, Bob led MicroTools, an engineering firm specializing in creating a variety of real-time embedded systems across a wide range of industries. Bob has been awarded 11 patents in many areas of embedded systems and motion control. Now retired, he enjoys building electronic projects with his grandchildren. You can reach him at Bob@ListeningToGod.org.

Additional materials from the author are available at:
www.circuitcellar.com/article-materials
References [1] to [10] as marked in the article can be found there.

RESOURCES

CodeMagic | www.codemagic.io

Expo | www.expo.dev

Flutter | www.flutter.dev

React Native | www.reactnative.dev

Start to Finish

Crystal Calibration with GPS Redux

By Stuart Ball

Stuart revisits an article from 2020 with a different approach, exploring the calibration of microcontroller clock crystals using GPS technology, aimed at enhancing timing accuracy in various applications. Key findings highlight the importance of calibration for long-term timing precision, detailing methods to achieve significant improvements in accuracy.

I described a method of calibrating a 32kHz clock crystal in a TM4C1233D5 microcontroller (MCU) with a GPS module in my article, "Calibrating an MCU's RTC Using GPS: Put Time on Your Side" (*Circuit Cellar* 360, July 2020) [1]. The Texas Instruments (TI) TM4C parts have an internal 32KHz oscillator for implementing a real-time clock (RTC). This new approach does the same thing but with a more generalized method that applies to MCUs without a dedicated RTC oscillator and calibration register.

CRYSTAL ACCURACY

A typical crystal, as used in an MCU, has some tolerance, measured in parts per million (ppm). So the frequency of a 14.7456MHz crystal, as I'm using in this design, will vary. It's exactly like the tolerance in a resistor or capacitor, but the tolerance of resistors and capacitors is measured in percent, or parts per hundred. Crystals are more accurate, so their tolerance is measured in ppm; 1ppm is equivalent to 0.0001%. Just like resistor/capacitor tolerance, the crystal frequency can be higher or lower than the nominal frequency.

To convert this to practical numbers, if the 14.7456MHz crystal I used in this design has a tolerance of ±30ppm, then the frequency variation is ±442Hz (14.7456MHz x 30/1,000,000). Therefore, the frequency of any given part may vary from 14.745158MHz (14,7456,000 – 442) to 14.746042MHz (14.456,000 + 442).

WHY CALIBRATE?

Why does any of this matter? In most applications, it doesn't. Crystal accuracy is usually close enough. However, you need calibration when any timing error is cumulative. Say you are using an MCU timer to divide the 14.7456MHz clock by 14,400 to produce a 1,024Hz interrupt. If the crystal is operating at the low end of the tolerance range (14.745158MHz), then over the course of a minute there will be 61,438 interrupts. With the nominal 14.7456MHz crystal frequency, there will be a total of 61,440 interrupts; the lower frequency produces two fewer interrupts in 1 minute.

This difference matters if you are using the interrupts to time something over a long period. The obvious application is some kind of clock, such as an alarm clock. But other applications may require accurate long-term timekeeping, and you don't always have access to the Internet to verify the time. Think of the controller for a sprinkler system, or a remote monitoring station, or anything that needs accurate time of day but can't access an external reference. If you were using the aforementioned timer interrupt to keep track of time, then over a week the time would be off by about 20 seconds. In a month, the time is off by over a minute. In a year, more than 10 minutes. Therefore, if you need to maintain accurate time without an Internet connection, you must make the clock more accurate.

In some situations, you could incorporate a GPS receiver to track the time. But that isn't always an option, due to either cost or location. The equipment won't always be situated where a GPS signal is available, such as under a metal roof. The way quartz crystal wristwatches maintain accurate time is by adjusting the actual crystal frequency with a variable capacitor. That's always an option but requires an additional manufacturing step and an accurate frequency counter.

THE CIRCUIT

Figure 1 shows a schematic of the demo circuit. U1 is a Microchip ATMega324 MCU. J1 is the MCU programming header, J2 is a diagnostic/debug serial port connector, and J3 is a socket header for a GPS module.

The GPS module I used (**Figure 2**) comes with a separate patch antenna that is plugged into the module. I have a couple of these modules with different pinouts for the Rx and Tx serial data. W1 lets me switch between modules. The GPS module is there to demonstrate the concepts used here. In a real product, you would use an external GPS module and only connect it when calibrating the oscillator.

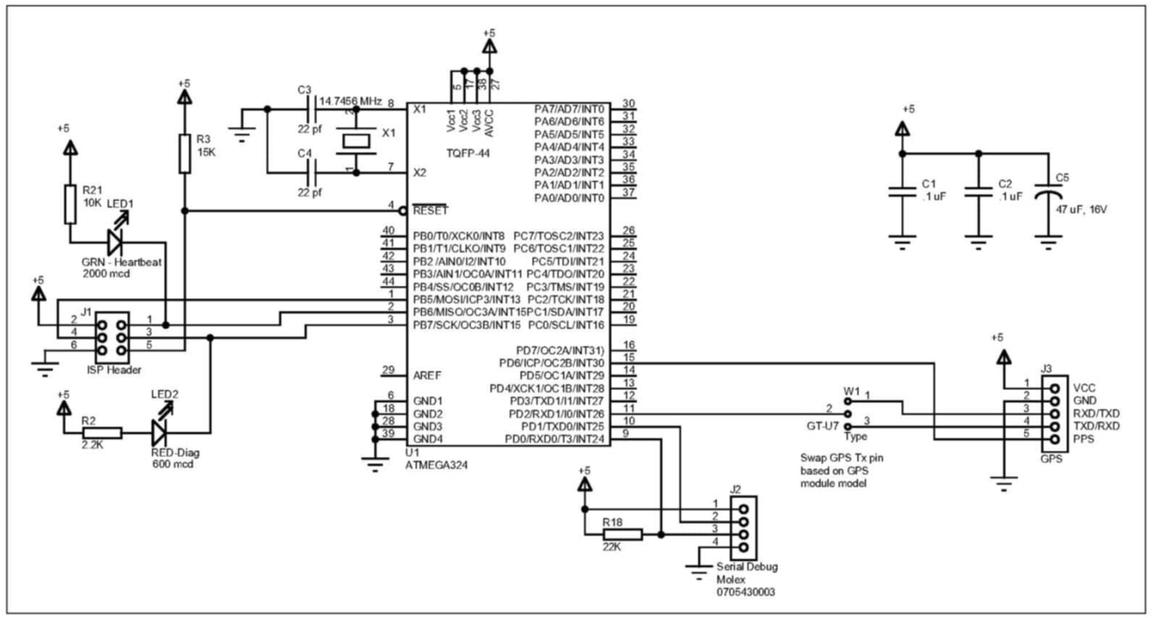


FIGURE 1
Schematic of the demo board.

Although the Rx pin from the GPS is connected to the MCU, it is not needed here; this project only uses the Pulse Per Second (PPS) signal from the GPS module. Once the GPS module synchronizes with the GPS satellites, it generates a very accurate pulse on the PPS line once per second. If you want to duplicate what I did here, be sure to get a GPS module with a 5-pin header. I've seen some that only provide 4 pins, omitting the PPS pin.

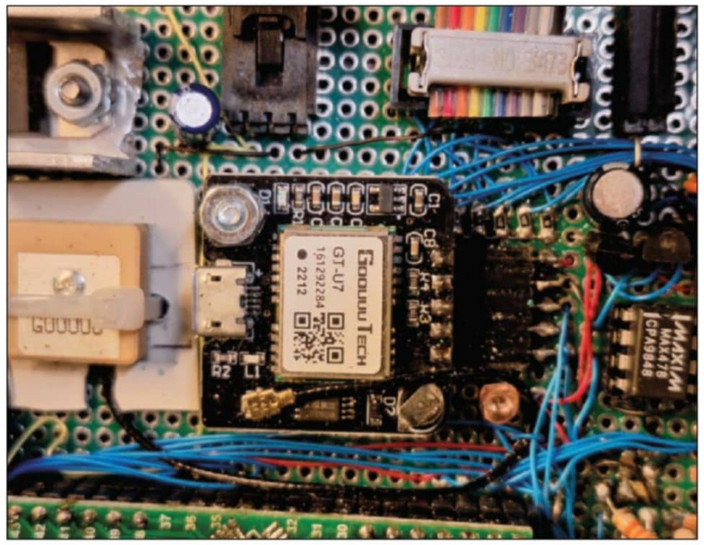


FIGURE 2
The GPS module used in the demo circuit.

COLUMNS

The PPS signal drives pin PD6, the input capture pin for T1, which triggers a capture of the 16-bit timer 1 (T1) count in the T1 input capture register (ICR). You could also use the analog comparator on PD3 to trigger a capture if PD6 was needed for something else.

The circuit operates at 5V simply because I needed a higher voltage for the original project. But you could operate it at 3.3V, as long as everything else works at that voltage.

Figure 3 is a schematic of the RS232 adapter to connect the debug port to a PC. If you've seen my articles using my standard debug/programming connector, you've seen this schematic. PCs don't have serial ports anymore, so this plugs into a USB-to-RS232 adapter. It cables to the demo board via a 4-pin cable.

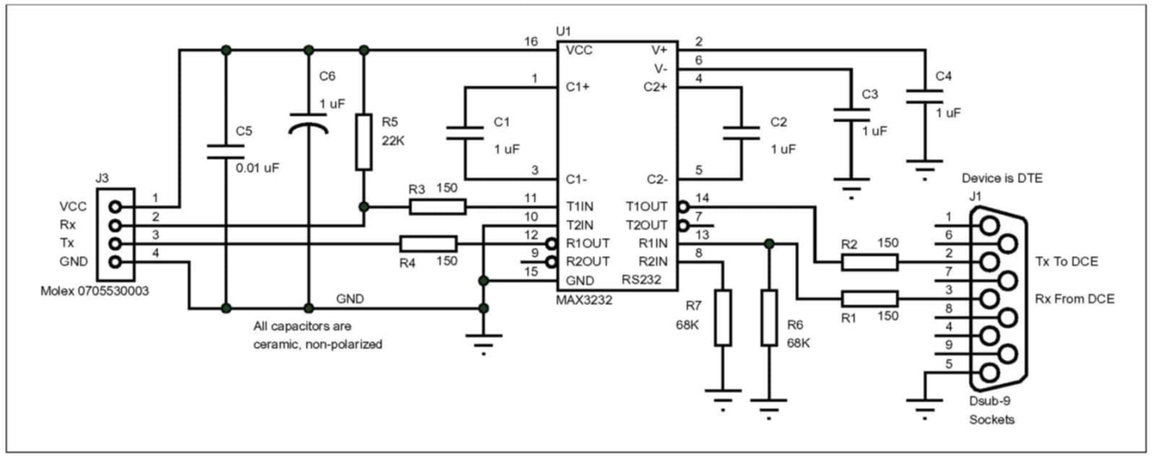


FIGURE 3
Schematic of the Interface circuit between the demo board and the PC.

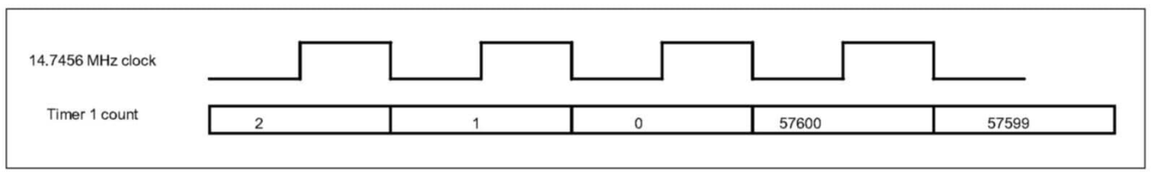


FIGURE 4
Timer 1 rollover and reload.

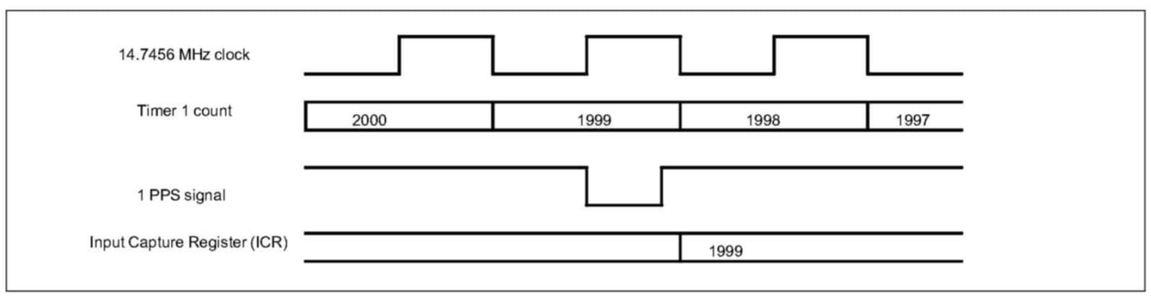


FIGURE 5
Timer 1 capture with the GPS 1 PPS signal.

TIMER 1

This example uses Timer 1 of the ATMega device. Other devices will have other timer options. The OCR1A register is loaded with the value 57,600. Each time that Timer 1 decrements to zero, it is reloaded with the value in the OCR1A register (**Figure 4**).

If OCR1A is modified with a different value, the next rollover will happen at that count value instead of 57600. This is the basis of the time calibration adjustment. The Timer 1 ICP (input capture) pin is connected to the 1 PPS signal from the GPS. Each time a 1 PPS signal occurs, the count value in Timer 1 is captured and stored in a register (**Figure 5**). The count value is just made up for this example.

THE SOFTWARE

In the original TMC1233 project, the RTC register would auto-update periodically, so once the correction value was entered, it simply had to be initialized, and it would run forever. Using the ATMega MCU requires an extra step. Each time the T1 timer rolls over (that is, gets reloaded with the value of 57,600), the software has to service that event. It can either be polled, or it can use an interrupt. I polled it in the demo code. The T1 routine increments a rollover counter to keep track of when 1 second has elapsed; 256 T1 rollovers is 1 second (assuming the crystal is exactly 14.7456MHz). When that happens, the OCR1 register is loaded with the corrected count (57,600 plus or minus the correction value, as needed). On the next rollover, the OCR1 register is loaded with the original 57,600 value. Thus, once per second the correction value replaces the normal divisor, and then the normal divisor is used in all other cases. This corrects for the accumulated error over the previous second.

When a GPS PPS pulse occurs, the software sends the elapsed time in seconds to the debug serial port, followed by the captured T1 count for that interval. The code also has a `#define` that causes the GPS time in HH:MM:SS to be added to the output. Elapsed time and captured T1 count are hex values; the GPS time is in ASCII decimal, as it is received from the GPS module. Here are a couple of example lines with GPS time included:

```
000C BE87 14:46:20
000D BE89 14:46:21
```

The GPS time is received from the GPS module on the module serial port. Since the GPS time isn't needed for this demo project, it's entirely optional, and the GPS serial port doesn't even need to be connected for the demo to work. Obviously, if the serial port isn't connected, there is no point in setting the flag to display the GPS time.

For this demo the software is all consolidated into one `main.c file`. Microchip Studio was used for development, although any AVR toolset should work.

HOW IT WORKS

Let's say that the crystal is running fast (frequency is higher than nominal) by 100ppm. This means that in one second, there will be 1474.56 more clock cycles than there would be if the frequency were exactly 14.7456MHz. T1, operating as a down-counter, will count 57,600 clocks then reload with 57,600 and start over. When the PPS signal from the GPS module occurs, the T1 count will be captured in the ICR. If the frequency was exactly nominal, T1 will be within 1 cycle of 57,600, so the count will always be the same, ±1. But in this case, it is off by 1,474 cycles per second, or 5.75 counts each 57,600-count interval. There are 256 of these intervals in a second. When the GPS PPS signal occurs, since the crystal is a little too fast, the error will have accumulated over a second, resulting in a T1 count error of 1,474. This is used as a calibration factor for adjusting the clock.

The actual calibration factor isn't the count itself. For one thing, you don't know where in the 1-second interval of the PPS signal the count started. So you have to subtract two successive count values to get the difference, which is the actual calibration value.

Once the calibration value is calculated, you can then insert it into the code and now the software will, every 256 seconds, adjust the 57,600 count value by the calibration value, adjusting the total counts for a 1-second interval to actually represent 1 second.

HOW TO CALIBRATE IT

The calibration process for the demo is as below:

1) Set the (`#define`) correction value to 0.
2) Capture the serial output of the elapsed time and captured count value for a few seconds.
3) Calculate the offset by subtracting successive counts.
4) Average a few values.
5) Insert the correction value into the code.

Below are 10 lines from the unadjusted output of the demo circuit:

```
0001   471B
0002   4643
0003   456C
0004   4495
0005   43BE
0006   42E6
0007   420F
0008   4138
0009   4061
000A   3F89
```

The left column is the elapsed GPS seconds, the right column is the captured values from the T1 timer. If you subtract the values in the

COLUMNS

right column from each other, you will see that the difference averages about 215. This works out to about 14.6ppm, as follows:

215 cycles per second/14.745MHz
= 14.58 cycles per million clock cycles, or 14.6ppm
This works out to 14.5µs error every second.

Since the value gets smaller on each sample, we know that the crystal is running a little slow; the captured count isn't keeping up with the PPS signal. The calibration value in the code was set to -215 (negative because the crystal is running slow). Here are the adjusted results:

```
0009   AA85
000A   AA86
000B   AA86
000C   AA87
000D   AA87
000E   AA87
000F   AA87
0010   AA88
0011   AA89
0012   AA89
```

It isn't quite exact, of course, because you can only adjust the count in integral values; no fractional values are allowed. But the original, unadjusted value would be off by 8.8 seconds per week (604,800 seconds/week x 14.5µs/second = 8.8 seconds). This is over 7 minutes per year. The corrected value, allowing for ±2 counts because the actual correction isn't an exact integer, will be 0.08 seconds in a week and a little over 4 seconds per year, or more than 100x better accuracy.

Once the crystal is calibrated, the GPS module and input capture pin are no longer needed. The internally stored correction factor will continue to be used. The input capture feature is only needed to capture the T1 count so that the calibration constant can be calculated. It would be entirely possible to write software that just does calibration and then use different software that uses the capture pin for something else entirely and doesn't even include the capture/display code. The demo software just demonstrates the capture and calibration method; the only parts that are needed in operating code are the parts that initialize timer T1, count T1 overflow events, and insert the calibrated crystal offset once per second to produce highly accurate time. Everything else could be easily discarded.

FREQUENCY VS. ACCURACY

The original 2020 article [1] focused on a quick way to calibrate a 32.768KHz clock crystal, using the TM4C dedicated RTC timer. The ATMega324, as well as a number of other MCUs, can operate from a 3KHz clock crystal; this is typically done to reduce power in a battery-operated design. This technique will also work with other crystals, including a 32kHz crystal, with some caveats.

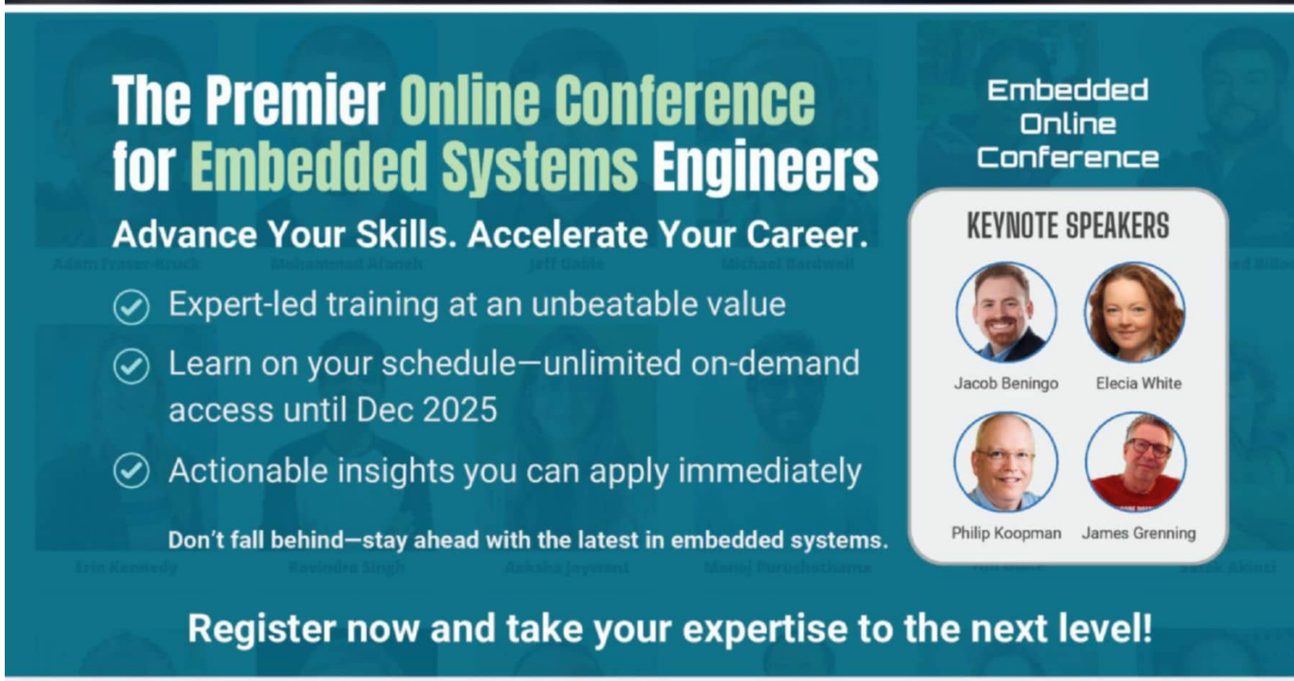
If a 32.768 crystal has a 100ppm tolerance, then the frequency ranges from 32764.7Hz to 32771.3Hz. 100ppm is a difference of only 3.27 cycles per second for the 32kHz crystal. The problem is that at 32kHz, you can't accumulate enough cycles in a single second to get a good calibration value. To compensate for this, the TM4C RTC performs a correction every 64 seconds, so that more clock cycles are collected and the correction is more accurate.

The 14.7456MHz crystal used in this demo is a high enough frequency that 1ppm variation is 14 clock cycles per second. A 7.3728MHz crystal would have 7 clock cycles at 1ppm variation. Both of those can achieve 1ppm accuracy with a 1-second measurement. But at frequencies below 1MHz, you will have to accumulate more than one second of clock cycles to measure and calibrate to 1ppm. Because of this, calibration will necessarily take longer. To get to 1ppm accuracy, you need to accumulate at least 1 million clock cycles, which is over 30 seconds for a 32kHz crystal. To calibrate any crystal operating below 1MHz, you will have to accumulate errors over multiple PPS events, and you will need at least two samples (about a full minute at 32KHz). In practice, you would want to accumulate several minutes and average the counts. The bottom line is that the time it takes to get a calibration value using this method depends on crystal frequency below 1MHz. Of course, you may not need 1ppm accuracy, and calibrating to 2ppm or 5ppm will take less time.

TEMPERATURE

Crystal frequency changes with temperature. The crystal is a piece of quartz of a specific size and shape. Like all materials, the size changes a little when temperature changes. There are two ppm values for a crystal—the tolerance and the temperature stability. The tolerance in ppm is the deviation from the specified frequency at some temperature, typically 25°C. So the example 14.7456MHz crystal described here

ABOUT THE AUTHOR
Stuart Ball recently retired from a 40+ year career as an electrical engineer and engineering manager. His most recent position was as a Principal Engineer at Seagate Technologies.


The Premier Online Conference for Embedded Systems Engineers
Advance Your Skills. Accelerate Your Career.
Expert-led training at an unbeatable value
Learn on your schedule—unlimited on-demand access until Dec 2025
Actionable insights you can apply immediately
Don't fall behind—stay ahead with the latest in embedded systems.
Embedded Online Conference
KEYNOTE SPEAKERS
Jacob Beningo
Elecia White
Philip Koopman
James Grenning
Register now and take your expertise to the next level!
Learn More: →
www.embeddedonlineconference.com

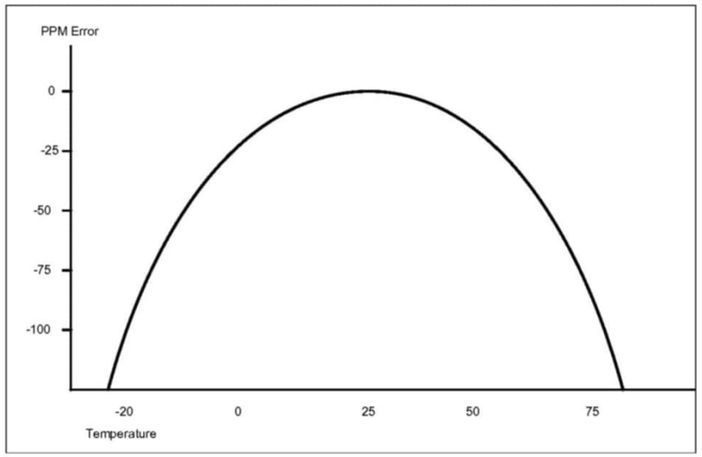


FIGURE 6
Crystal temperature variation.

with a 30ppm tolerance will vary no more than 30ppm from the specified frequency at 25°C.

Frequency stability is a different value and is specified over some temperature range such as -10°C to 60°C (**Figure 6**). The 30ppm crystal might have a stability of 50ppm over -10°C to 60°C. This means that while the tolerance at 25°C is 30ppm, the frequency can vary by as much as 50ppm over the full temperature range. The effect of tolerance and stability are additive, so that hypothetical crystal will have a total tolerance of 30ppm + 50ppm over the temperature range. It will still have a 30ppm tolerance at 25°C, but the further the temperature gets from 25°C, the more it will vary. Note that this is the maximum range; if the actual crystal is accurate to 10ppm, then the total tolerance plus stability will be 60ppm for that specific part.

Crystals come in different temperature ranges, so if your operating temperature is fairly wide (e.g., a clock in a car), you might want to use a crystal with better temperature stability. You'll pay for it, of course. Note that different cuts of crystals have different temperature variation curves; Figure 6 is just an example.

TEMPERATURE COMPENSATION

If you need better accuracy over a wider temperature range, you can calibrate the crystal for that as well. You would heat and/or cool the crystal (or the entire product) over the temperature range where it needs to operate and at each temperature step. Maybe you vary the temperature in 10°C steps. When you're done, you have a table of calibration constants versus temperature. Embed that in your MCU EEPROM and you know what calibration constant to use at any given temperature. Of course, the MCU has to be able to measure ambient temperature for this to work. Calibration will take longer as you have to step through the temperature range, allow it to stabilize, and then capture the timer deviation to build the table of calibration constants. This is something you would only want to do if you really need to high accuracy over a range of temperatures.

This might seem to be overkill since so many electronics operate at or near room temperature. But a product operating outdoors through all four seasons in a hot or cold climate may need temperature compensation.

COMMERCIALIZING IT

Of course you wouldn't want a GPS module installed in every production unit; that would defeat the purpose. What you could have is a master GPS module that drives multiple units at the final test stage of the product and add code to do the calculations and set the offset, storing it in the MCU EEPROM (instead of as a hard-coded `#define`). Each unit could self-calibrate in a few seconds.

OTHER MCUS

You could implement this on different MCUs; the key features are a timer that has auto-reload of a divisor value and an ability to capture the count value in an internal register in response to an external signal. A number of the ATMega parts have this capability. The Microchip XMega parts also support this. In addition, some of the TI TM4C1233 parts have this capability.

This is far from an exhaustive list of MCUs that can implement this technique. If you have a specific MCU family that you want to try, you'll have to look through the datasheet to see if the necessary timer features are supported.

CONCLUSION

Measuring accurate time is sometimes important. You can't always connect to a Wi-Fi network, and you can't always embed a GPS module in your product. In my last article on this subject, I explained a way to quickly calibrate the built-in RTC in the TI TM4C family of parts. Here, I've demonstrated how to apply the same basic technique to a wider expanse of MCUs, using general-purpose timers. If you are in a situation where you need this kind of long-term timing accuracy, I hope this is helpful.

Additional materials from the author are available at:
www.circuitcellar.com/article-materials
References [1] as marked in the article can be found there.

RESOURCES

Microchip | www.microchip.com

Texas Instruments | www.ti.com

PRODUCT NEWS *By Kirsten Campbell*

Cincoze Launches 15.6" FHD Panel PC and Monitor for Industrial and Outdoor Applications

Rugged embedded computer brand—Cincoze has added a new 15.6" full high definition (FHD) display option for HMI applications in harsh environments to its Display Computing—CRYSTAL product line. Almost 20 new products have been launched at this time, including rugged industrial panel PCs (CV-W115FH/P) and monitors (CV-W115FH/M) suitable for general harsh environments, as well as sunlight readable panel PCs (CS-W115FH/P) and monitors (CS-W115FH/M) designed for high-brightness outdoor environments, providing even more comprehensive options across the CRYSTAL product line.

The Cincoze 15.6" industrial panel PC (CV-W115FH/P) and monitor series (CV-W115FH/M) offer exceptional picture quality for industrial applications such as displaying complex SCADA data or image monitoring. This is thanks to its FHD (1920×1080) resolution, 800:1 contrast ratio, and 16.7 million color display, which, when combined with resistive or capacitive touch and a wide 178° viewing angle, provide the optimal user experience. The backlight life of up to 50,000 hours ensures long-term stable operation. The panel PC series has various performance options, including Intel, Atom, Pentium, or Core to suit specific use cases.

The Cincoze 15.6" sunlight readable panel PC (CS-W115FH/P) and monitor (CS-W115FH/M) series are built for harsh outdoor environments, with a 1,200 nits high brightness screen ensuring readability even under direct sunlight outdoors. The series is built to withstand harsh conditions and features an IP65 waterproof and dustproof front panel, a shock-resistant aluminum die-cast front bezel, and a 7H hardness touch surface. They support wide temperatures (-20°C to 70°C) and wide voltages (9V to 48V DC), and include overcurrent, overvoltage, and electrostatic discharge (ESD) protection. The series is UL-certified and complies with the EN62368-1 standard, making it ideal for applications such as outdoor interactive information stations, charging piles, and automatic ticket vending machines.

The series utilizes Cincoze's exclusive Convertible Display System (CDS) technology for flexible configuration with computer modules (P2000/P1000 series) or monitor modules (M1000 series). This enables the creation of industrial or sunlight readable panel PCs and monitors suitable for industrial environments or outdoor use. The plug-and-play design means only a single module requires replacement during on-site maintenance, effectively shortening downtime, reducing maintenance costs, and facilitating future upgrades.

Cincoze | www.cincoze.com/en

Contrinex Launches Smart Photoelectric Sensors

Contrinex has introduced smart versions of its proven C23 and M18 photoelectric sensor series, enhancing efficiency in manufacturing. These high-resolution smart sensors offer a detection range from 5mm to 1,200mm, feature digital measurement and switching capabilities, and are fully programmable. Their robust stainless steel housing ensures reliable operation even in harsh environments.

The newly launched smart photoelectric sensors are available in two versions: SMART BASIC and SMART DMS. Operating with a focused red light beam (630nm/640nm), they offer varying detection ranges: 10mm to 300mm (SMART BASIC with background suppression, ≤500Hz), 5mm to 1,200mm (SMART BASIC retro-reflective sensor, ≤1000Hz), and 10mm to 200mm (SMART DMS measuring sensor, ≤500Hz).

The SMART BASIC series provides digital teach and switching point configuration, advanced switching logic, configurable alarm thresholds and sources, reserved memory for user-defined data tags, an external SIO input, and a wire-teach function. Meanwhile, the SMART DMS measuring sensors offer extended data acquisition, including statistical calculations, and a calibrated and precisely linearized digital measurement output. Their advanced switching logic allows for a broader range of applications compared to standard models. A key feature of these sensors is their state-dependent self-monitoring capability, which helps reduce maintenance costs.

Contrinex's photoelectric sensors are known for their exceptional detection capabilities and reliable background suppression. Thanks to their patented measurement principle, they can accurately detect even thin, transparent objects at close range. Additionally, they offer high repeat accuracy and reliability, even in challenging environments (IP68/IP69K). These sensors are ideal for industries such as warehouse logistics, automated manufacturing, food processing, and semiconductor production.

Contrinex | contrinex.com

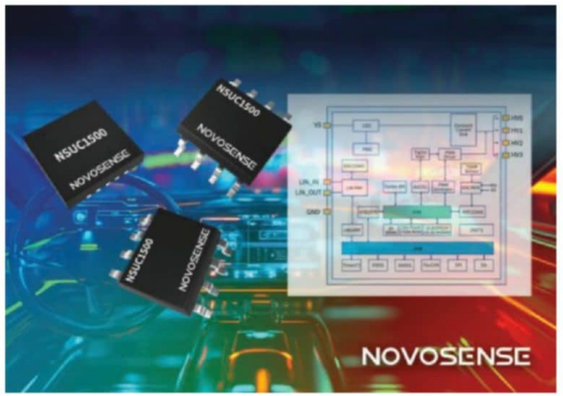
NEW PRODUCT SUBMISSIONS— E-mail: product-editor@circuitcellar.com

NOVOSENSE Introduces Highly Integrated SoC for Automotive Lighting Systems

NOVOSENSE Microelectronics, a semiconductor company specializing in high-performance analog and mixed-signal chips, has announced the launch of the NSUC1500-Q1, a highly integrated system-on-chip (SoC) designed to meet the demands of modern automotive ambient lighting systems.

The NSUC1500-Q1 can be combined with the high-precision NSLxxxxx series of linear LED drivers optimized for conventional taillights, through-type dynamic taillights, and illuminated grilles for an integrated automotive lighting solution.

The integrated solution is targeted at automotive taillights and interior lighting modules, EV charging door indicators, and other RGB/RGBW-based lighting systems requiring compact, efficient, and intelligent control.



"As the demand for interior ambient lighting increases, OEMs face ongoing challenges in delivering accurate dimming, color mixing control and thermal reliability, especially across complex distributed networks," said Feng GAO, Product Line Marketing Manager of NOVOSENSE. "The NSUC1500-Q1 reduces system complexity down to a single chip, saving PCB space and lowering overall BOM cost for automotive lighting module developers."

At the heart of the NSUC1500-Q1 is a 32-bit ARM Cortex-M3 core with 32KB Flash, 2KB EEPROM, 15K ROM, and dedicated memory for runtime operations. The SoC incorporates four LED driver channels capable of delivering up to 64mA each, controlled through enhanced 16-bit independent PWM outputs. A 12-bit ADC, in combination with an integrated temperature sensor and voltage monitoring, enables automatic compensation for LED color shift due to temperature changes or aging.

The integrated LIN transceiver and controller support LIN 2x and SAE J2602 communication standards, with features such as auto-addressing to simplify network setup. The NSUC1500-Q1 also supports software updates via the ROM-based UDS bootloader without occupying Flash memory.

The NSUC1500-Q1 is available in QFN-24, SOP8, and HSOP8 packages and supports a wide operating voltage range from 6V to 28V.

NOVOSENSE | www.novosns.com/en

SemiQ Announces 1200V SiC MOSFETs Six-Pack Modules with High Power Density and Low Switching Losses Enables Compact, Cost-Optimized Systems

SemiQ Inc, a designer, developer, and global supplier of superior silicon carbide (SiC) solutions for ultra-efficient, high-performance, and high-voltage applications, has announced a series of highly efficient 1200V SiC MOSFET Six-Pack Modules. These have been designed to enable lower cost and more compact system-level designs at large scale.

The rugged, high-speed switching SiC MOSFETs implement a planar technology with rugged gate oxide and feature a reliable body diode. These are arranged in a three-phase bridge topology, with the modules additionally featuring split DC negative terminals, press-fit terminal connections and a Kelvin reference for stable operation.

The high-power-density modules benefit from low switching losses, as well as low junction-to-case thermal resistance and all parts have been tested beyond 1350V, with 100% wafer-level burn in (WLBI).

They have been developed for applications including AC/DC converters, energy storage systems, battery charging, motor drives and PFC boost converters, including EV fast charging, induction heating and welding, renewable energy supplies and UPS.

The modules are operational to 175°C junction temperature, and have been designed for easy mounting, including direct mounting to a heatsink. The product family has been launched with 20mΩ, 40mΩ, and 80mΩ variants (GCMX020A120B2T1P, GCMX040A120B2T1P, and GCMX080A120B2T1P) that have a power dissipation of 263W, 160W, and 103W, respectively.

They conduct a continuous drain current of 29A to 30A*, and a pulsed drain current of 70A*. Additionally, they have turn-on switching energy of 0.1mJ to 0.54mJ* and a turn-off switching energy of 0.02mJ to 0.11mJ*, with a switching time of 56ns to 105ns*.

SemiQ Inc | semiq.com

Ironwood Electronics Presents ATE Test Contactor for MSOP10 with Manual Actuator

Ironwood Electronics introduces its automatic test equipment (ATE) test contactor, incorporating Pico Raptor 2 solid contact technology with elastimers. This contactor is specifically engineered to handle MSOP devices, precisely those measuring 3mmx3mm with 10 leads, featuring a 0.5mm pitch. The physical assembly of this test contactor includes a socket body frame, an alignment plate, and notably, a replaceable cartridge. This integrated design ensures compatibility and reliable contact for the specified device package.

A key innovation designed to address embedded engineers' needs for efficiency and minimal downtime is the replaceable cartridge system. The source highlights a significant advantage: once testing is completed, or after a substantial number of devices have been processed—specified potentially around 500,000 devices—the cartridge is designed for easy removal and replacement. This simple process translates directly to minimal production downtime, a critical factor in high-volume testing environments where contactor wear is a primary concern for throughput.

Beyond high-volume production, the contactor system is also designed to facilitate crucial engineering tasks like initial debug and verification. To prepare the socket for use, it is first mounted onto a board, as demonstrated in the source with a clear acrylic plate. This provides a stable base for both manual and automated operations.

For the important initial debug and verification phase, a specific setup is employed. A plate is assembled specifically for a manual lid. The device under test is then manually placed into the socket. Following device placement, the manual actuator is positioned onto the device. Pressure is then applied by turning a screw, ensuring proper contact for debugging purposes. This allows engineers to focus on clearing issues before moving to automated testing. Once the necessary debug is complete, the manual actuator and its associated plate are removed.

The system then seamlessly transitions to high-speed production testing. In this phase, devices are typically handled by an automated handler, such as the Esmo Talos handler mentioned in the source. The handler is responsible for feeding the devices directly into the pocket of the contactor. This automated process enables the system to perform tests in a very fast production manner, optimizing throughput after the initial design and debug challenges have been overcome using the manual setup.

Ironwood Electronics
ironwoodelectronics.com

e-con Systems Latest 4K 140dB HDR Front View Camera Delivering Superior Long-range Imaging

Introducing the STURDeCAM88, a 4K HDR forward-facing Camera engineered to overcome these barriers and provide precise, real-time vision in any condition. This camera is specifically designed for superior, reliable imaging at long distances in various mobility applications. It integrates e-con Systems' industry-leading ISP tuning expertise with OmniVision's automotive-grade OX08B40 sensor.

This powerful combination delivers ultra-clear 4K imaging coupled with an impressive 140dB High Dynamic Range (HDR). The high dynamic range provides the essential clarity needed to detect obstacles, pedestrians, and lane markings for ADAS systems, even in the most challenging lighting environments.

For robust traffic sign and light recognition, the camera incorporates an LED Flicker Mitigation (LFM) feature. This is crucial for eliminating interference caused by flickering LED lights, ensuring accurate and reliable recognition.

Durability and reliable operation in harsh environments are ensured as the STURDeCAM88 is housed in a rugged, IP67-rated enclosure. Furthermore, to guarantee resilient, uninterrupted streaming, the camera includes a Health Monitor and Recovery feature that actively tracks the health of the sensor, ISP, and data link in real-time.

Integration into system designs is streamlined as the camera is compatible with all NVIDIA Jetson image processing platforms. For advanced vision needs requiring multiple viewpoints, e-con Systems offers solutions supporting up to eight cameras on the NVIDIA Jetson platform using GMSL2 connectivity. GMSL2 enables high-speed, long-distance data transmission, essential for complex mobility systems.

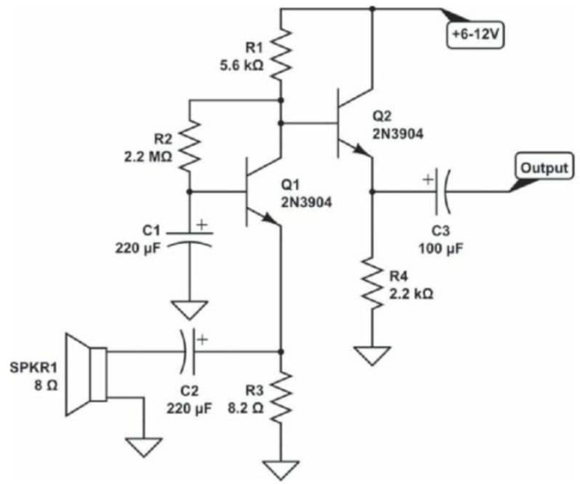
The STURDeCAM88 represents a significant advancement, powering mobility with superior vision beyond traditional limits.

e-con Systems | e-consystems.com

TEST YOUR EQ

Contributed by David Tweed

Answer 1 —
This configuration is known as a common-base amplifier. The input signal is applied to the emitter, and the output is taken from the collector of the transistor. The base is held at a constant voltage by the combination of R2 and C1, which have a time constant of 484 seconds.



Answer 2 —
The common-base configuration provides only unity current gain, but a large amount of voltage gain. Since the speaker is a low-impedance source, it operates at relatively high currents and low voltages. This arrangement of the transistor acts to impose the AC current of the speaker across the much larger load resistor connected to the collector, providing voltage gain equal to the ratio of the impedances.

Answer 3 —
The AC voltage across the B-E junction is on the order of 1mV or less, and the nonlinearity of this junction has little effect on circuit performance. The base current is a tiny fraction (1/beta) of the emitter current, and any distortion of this current is merely added to the collector current, not amplified by the transistor.

Answer 4 —
The biggest problem with this circuit is the fact that the impedance of the speaker varies quite a bit with frequency, which makes the voltage gain of the circuit vary as well. The low value of emitter resistor R3 helps to limit this effect.

TEST YOUR EQ

HIGH PERFORMANCE, HIGHLY SCALABLE 30V-75V PORTFOLIO OF BUCK CONVERTERS

Optimized to power the highest precision loads with excellent accuracy

- Adjustable V_{OUT} from 0.8V to ~90% of V_{IN}
- ± 1% FB voltage accuracy over -40°C to 125°C
- Default 400kHz or adjustable switching frequency from 200kHz to 800kHz
- Available in 20ld QFN and 16ld HTSSOP packages

Visit **renesas.com/buck-integrated-fets** to learn more

RENESAS

IDEA BOX

The Directory of PRODUCTS & SERVICES

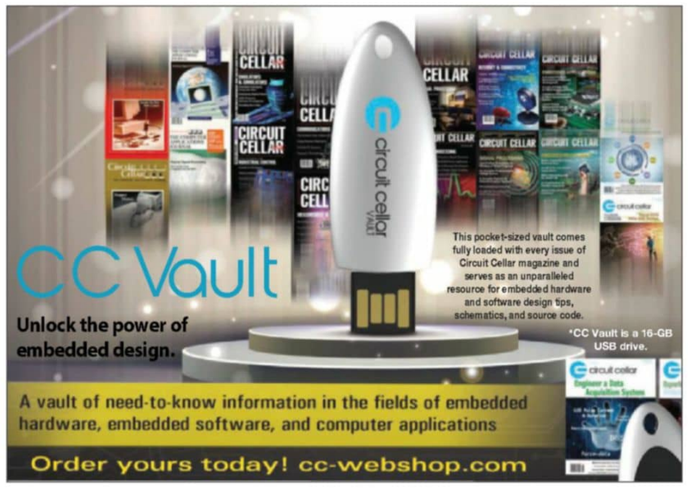
AD FORMAT:
Advertisers must furnish digital files that meet our specifications (www.circuitcellar.com/mediakit).

All text and other elements MUST fit within a 2" x 3" format.
E-mail adcopy@circuitcellar.com with your file.

For current rates, deadlines, and more information contact Hugh Heinsohn at 757-525-3677 or Hugh@circuitcellar.com

When it comes to robotics, the future is now!
Advanced Control Robotics simplifies the theory and best practices of advanced robot technologies, making it ideal reading for beginners and experts alike.
With this book, you'll learn about:
• Communication
• Technologies
• Control Robotics
• Embedded Technology
• Programming Language
• Visual Debugging... and more
ADVANCED CONTROL ROBOTICS
HANNO SANDER
Get it today, cc-webshop.com

RAPID18
Universal Development Kit for PIC18 Family
ONLY $99!
• Real-time Clock/Calendar with Supercap
• USB port for text communication to the running program
• Built-in bootloader for loading code
• Excellent for AC/DC Data Logging
Kit includes everything you need to develop with a PIC® MCU:
• Rapid 18 Prototyping Board
• Single-Chip IDE C Compiler
• Exercise Book
• Power Adapters and Cables
sales@ccsinfo.com (262) 522-6500 x35
www.ccsinfo.com/cc1223


CC Vault
Unlock the power of embedded design.
This pocket-sized vault comes fully loaded with every issue of Circuit Cellar magazine and serves as an unparalleled resource for embedded hardware and software design tips, schematics, and source code.
*CC Vault is a 16-GB USB drive.
A vault of need-to-know information in the fields of embedded hardware, embedded software, and computer applications
Order yours today! cc-webshop.com

embeddedTS
embeddedTS.com
TS-7100-Z
Powered by the high performance, ultra-efficient Arm® Cortex®-A7 based i.MX6UL CPU
...In an enclosure as small as two decks of cards
(2.4" x 3.6" x 1.7")
Best In Show
Embedded Computer Design
Made in the USA
with global parts

The Future of Graphical User Interfaces

Efficient GUI Design

Experience Flow with Microchip Graphics Suite (MGS)

By
Bill Li,
Graphics and Vision Applications Manager, Microchip Technology's MPU32 Business Unit

In today's embedded systems landscape, designing compelling and user-friendly Graphical User Interfaces (GUIs) is not a luxury, it's a necessity. From consumer electronics and industrial machinery to healthcare devices and aerospace control panels, users expect responsive, visually engaging interfaces that enhance their experience.

Achieving these goals can be challenging for both UI/UX designers and embedded engineers, both of which are critical to creating modern GUIs. Typically, these two distinct groups of creators operate in vastly different domains. The former is focused on visual aesthetics while the latter is more attentive to functionality and hardware integration.

Tools that seamlessly serve both creator groups minimize the complexity of the development process and accelerate product time-to-market. Microchip Technology developed Microchip Graphics Suite (MGS), a comprehensive solution to bridge this gap, enabling a more efficient GUI design experience. The result is a streamlined workflow where static design elements and dynamic interactive features come together seamlessly in the final product.

This article explores how the MGS Client Experience Flow, illustrated in **Figure 1**, improves GUI development for embedded systems. We will focus on how MGS Composer and MGS Simulator work together to empower designers and engineers, enabling them to create polished, user-friendly GUIs more quickly and efficiently than ever before.

THE IMPORTANCE OF A UNIFIED GUI DEVELOPMENT WORKFLOW

Developing a functional and visually appealing GUI traditionally required a lengthy, iterative process involving multiple teams working independently. UI/UX designers would create the look and feel of the interface using photo editing design tools that produced static designs, which were then passed on to embedded engineers. Engineers would translate these designs into working software, which often resulted in discrepancies between the intended design and the final product due to communication breakdowns, differences in interpretation, or tool limitations.

This fragmented workflow causes development delays and an overall lower-quality GUI design. MGS was developed to address these challenges to provide an integrated workflow that improves collaboration between designers and engineers while streamlining the overall GUI development process.

MGS CLIENT EXPERIENCE FLOW: A STEP-BY-STEP BREAKDOWN

The MGS Client Experience Flow illustrates how MGS facilitates a modern, integrated design workflow that accelerates the development of embedded GUIs. Let's break down each step of the process:

Step 1: UI/UX Designer Creates the Visual Layout

The design process begins with the UI/UX designer, who uses standard design tools to create the visual layout of the GUI. This includes defining key elements such as buttons, icons, and backgrounds—components that shape the overall aesthetic of the interface. These visual designs establish the foundation for the GUI's structure and functionality, representing the intended look and feel of the final product.

Step 2: MGS Composer—Bringing Visuals to Life

Once the visual design is ready, it is transferred to MGS Composer, the core tool within the MGS for developing and managing GUIs. MGS Composer allows both designers and engineers to manipulate and refine the interface in a single environment.

The tool provides an intuitive visual interface where static elements such as images, fonts, and buttons can be easily arranged and structured for deployment on an embedded system. MGS Composer automatically generates the C code necessary to implement the graphical design, reducing the amount of manual coding required by the engineer.

By automating the code generation process, MGS Composer ensures that the GUI is correctly formatted for the hardware platform, saving time and reducing the chance of errors. This integration streamlines the workflow, allowing designers and engineers to focus on what they do best without the need for excessive back-and-forth communication.

TESTING AND SIMULATION: VERIFYING THE DESIGN

A critical phase in GUI development is testing and verifying that the design behaves as expected under real-world conditions. In traditional workflows, engineers often had to wait for the final hardware to become available before thoroughly testing the GUI's functionality. However, MGS offers powerful simulation tools that allow for thorough testing even before hardware is in hand.

Step 3: Web Mode Simulation—Sharable Reviewing

Using the MGS Simulator, developers can run a complete web-based demo of the GUI, allowing them to interact with the interface in a browser environment. This is a valuable step for testing fundamental interactions and verifying that the design functions as intended. Furthermore, stakeholders and clients can review the web demo and provide early feedback on the design without the need for physical access to hardware.

The MGS Simulator also enables in-depth testing, including debugging graphical elements and interactions in a desktop environment. This hardware-independent testing phase accelerates development, allowing issues to be caught and resolved early in the process, minimizing the risk of last-minute changes or costly rework.

Step 4: Native Mode Simulation—Debugging on Desktops

In native mode, the MGS Simulator generates the necessary files to enable desktop simulations within an Integrated Development

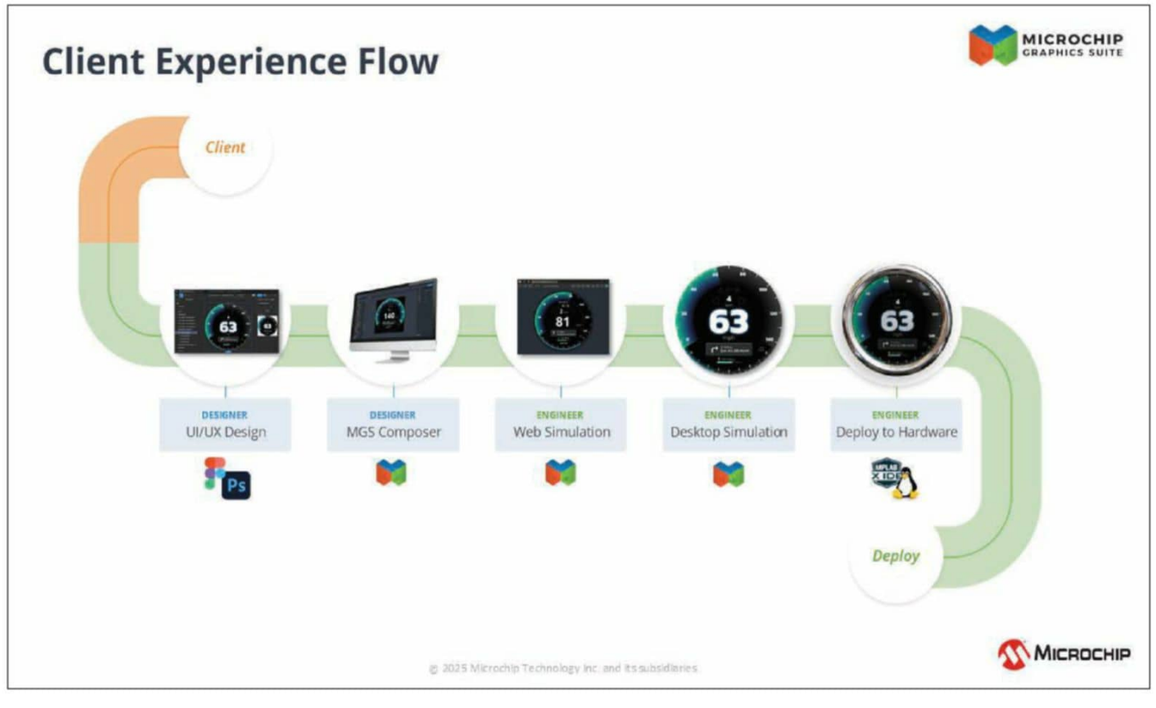


FIGURE 1
MGS Client Experience Flow demonstrates how the entire process can be executed more efficiently, with fewer opportunities for miscommunication or error.

Environment (IDE) such as Visual Studio Code (VS Code). This allows engineers to test the GUI design in a controlled environment, which is crucial for fine-tuning performance, memory usage, and overall system behavior.

By enabling detailed debugging at this stage, MGS helps engineers optimize the GUI's performance to meet the constraints of the target hardware platform. Whether the system has limited processing power or memory, engineers can make the necessary adjustments before deploying the design on physical hardware.

FINAL IMPLEMENTATION: DEPLOYING ON HARDWARE

Once the design has been refined and tested through simulation, the final step is deploying it on the actual hardware platform. The MGS Client Experience Flow ensures that this process is smooth, predictable, and free from significant issues.

Step 5: UI Design on Hardware

With the code generated by MGS Composer and debugged through MPLAB X (using the Harmony v3 environment or GNU Debugger (GDB) in the case of embedded Linux environments), the embedded engineer can deploy the GUI on the target hardware platform. Whether the system runs on Microchip's 32-bit microcontrollers (MCUs) or microprocessors (MPUs), the design will run seamlessly, thanks to the efficient integration of MGS into the development pipeline.

This final step allows the client to review the fully functional UI on hardware, providing them with a tangible product to interact with and ensuring that the design meets their expectations before it moves to production.

CONCLUSION: A SEAMLESS GUI DESIGN EXPERIENCE WITH MGS

The MGS represents a significant leap forward in the way UI/UX designers and embedded engineers collaborate on embedded GUI development. By providing a unified toolset that simplifies everything from design to deployment, MGS ensures that the visual intent of the GUI is preserved while the interface is optimized for hardware performance.

The MGS Client Experience Flow demonstrates how the entire process—from initial design through testing and final deployment—can be executed more efficiently, with fewer opportunities for miscommunication or error. MGS Composer and MGS Simulator work together to provide a complete solution that reduces development time and ensures that the GUI performs as expected, even on resource-constrained embedded systems.

In an era where time-to-market is critical, having an integrated workflow that bridges the gap between visual design and technical implementation is invaluable. For anyone working on embedded GUI development, the Microchip Graphics Suite offers the tools necessary to create responsive, visually appealing, and efficient interfaces that enhance the user experience.

By adopting MGS, development teams can significantly improve their design process, ensuring faster project completion, reduced errors, and a more enjoyable collaboration between designers and engineers.

TECH THE FUTURE

RESOURCES

Microchip | www.microchip.com

ABOUT THE AUTHOR

Bill Li is the Applications Engineering Manager in the MPU32 business unit at Microchip Technology Inc. With more than 25 years of experience in embedded software development, he brings a broad and versatile background spanning telecommunications, industrial lighting control, and video game development. This diverse industry expertise enables him to lead the team behind the industry-leading Microchip Graphics Suite. Bill holds a Bachelor of Science in Electrical Engineering from Queen's University in Kingston, Ontario, and an MBA from the University of Arizona Global Campus.

SeaFire Micros, Inc.
IP & Technology for Cloud Networks
Architecture
Protocol
SW FW i/f
hardware
L1-L7
TV
Phone
Data Center
Web
WiFi
Servers
PC Users
Design through Production
Military — Commercial
Hardware (Analog/Digital)
Firmware • Software • FPGA
Patented Network Technology | Unique Cloud Technology
No Compete Government Procurement
No Compete Government Funding
20% DISCOUNT on 1st Purchase
info@seafire.com | seafire.com

TS-9370

Low Power, High Performance SBC

The TS-9370 is a versatile and power-efficient industrial SBC built for rugged applications. Powered by the NXP i.MX9352 64-bit ARM® architecture plus two Cortex®-A53 processors operating up to 1.7 GHz, it delivers robust performance with low power consumption.

32 GB eMMC (upgradable)

2 GByte LPDDR4 RAM

MikroBUS socket
For MikroE Click Boards

Made in the USA
with global parts

VISIT US THIS JUNE IN SANTA CLARA, CALIFORNIA AT SENSORS CONVERGE BOOTH #742

embeddedTS

TS-4100

Ultra-Efficient and Fully Featured

The TS-4100 features an NXP i.MX 6UltraLite CPU, industrial temp range, FPGA, eMMC, and customizable I/O. Comes preloaded with Debian Stretch.

TS-7100-Z

As Small as Two Decks of Cards

Powered by an Arm® Cortex®-A7 i.MX6ul CPU, it runs efficiently at <2W, featuring Dual Ethernet, Bluetooth, WiFi, and NimbeLink/Digi sockets.

www.embeddedTS.com